VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Interaktivní výuka lineární algebry na internetu

Interactive learning of linear algebra on internet

2009 Lukáš Fiala

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě …… ……

Lukáš Fiala

Poděkování

Touto cestou bych rád poděkoval vedoucímu této bakalářské práce, panu Doc. Mgr. Vítu Vondrákovi, Ph.D., za odbornou pomoc při vedení mé práce.

Abstrakt

Tato bakalářská práce se zabývá usnadněním výuky lineární algebry ve vybraných oblastech. Cílem práce je navržení a vytvoření webové stránky, která uživatelům pomůže snázeji dosáhnout požadované znalostní úrovně.

Práce je rozdělena do dvou hlavních částí. První část je tvořena java applety, ve kterých je možno si prakticky vyzkoušet zejména práci s maticemi a řešení soustav lineárních rovnic, různé rozklady a kongruenci symetrických a diagonálních matic. Druhá část obsahuje kontrolní testy pokrývající vybrané kapitoly lineární algebry.

Klíčová slova

Java Applet, MathML, HTML, lineární algebra, matice, WWW

Abstract

This bachelor project deals with the simplification of the education of linear algebra in the selected sections. The goal of my work is to design and create a web page which would help users to reach required knowledge level more easily.

The project is divided into two major parts. First of them is created by java applets in which is possible to exercise working with matrices and solving the system of linear equations, different sorts of matrix factorizations and congruence of symmetric and diagonal matrices. The second part includes control tests covering chapters of linear algebra.

Keywords

Java Applet, Mathematical MarkupLanguage, HyperText Markup Language, linear algebra, matrix, World Wide Web

Seznam použitých symbolů a zkratek

.NET	soubor technologií v softwarových produktech, které tvoří celou platformu
API	rozhraní pro programování aplikací (Application Programming Interface)
ASCII	kódová tabulka definující znaky (American Standard Code for Information Interchange)
ASP.NET	součást .NET frameworku webových aplikací a služeb (Active Server Pages)
CSS	jazyk pro formátování internetových stránek (Cascading Style Sheets)
DOM	objektově orientovaná reprezentace XML nebo HTML dokumentu (Document Object Model)
DVI	formát dokumentů vzniklých pomocí překladačů typu TeX (DeVice Independent)
GIF	grafický formát určen pro rastrovou grafiku(Graphics Interchange Format)
HTML	značkovací jazyk pro hypertext (HyperText Markup Language)
IBM	společnost v informačních technologiích (International Business Machines Corporation)
ID	identifikace (Identification)
JPG	formát obrázků ve fotorealistické kvalitě (Joint Photographic Experts Group)
MathML	značkovací jazyk pro matematické symboly (Mathematical MarkupLanguage)
MSIE	grafický webový prohlížeč (Microsoft Internet Explorer)
PDF	souborový formát pro ukládání dokumentů (Portable Document Format)
PHP	skriptovací programovací jazyk (Personal Home Page)
PNG	formát pro bezeztrátovou kompresi rastrové grafiky (Portable Network Graphics)
PS	jazyk pro grafický popis tisknutelných dokumentů (Postscript)
TEX	sázecí systém
W3C	mezinárodní konsorcium - vyvíjení webových standardů
WWW	celosvětová síť propojených hypertextových dokumentů (World Wide Web)
XHTML	hypertextový značkovací jazyk (eXtensible HyperText Markup Language)
XML	obecný značkovací jazyk (eXtensible Markup Language)

Obsah

1 Úvod

Cílem interaktivní výuky lineární algebry na internetu bylo vytvořit webovou stránku, která bude plnit funkci praktického procvičení vybraných problémů lineární algebry. Návrhem bylo vhodným způsobem nabídnout uživateli možnost prakticky vyzkoušet své teoretické znalosti. Šlo tedy o vytvoření pomocných nástrojů, které by splňovaly část výukovou (pomocné nástroje pro procvičení Gaussovy eliminace, Gauss-Jordanovy eliminace, LU rozkladu, výpočtu inverzní matice, kongruence symetrické matice, LDL^T rozkladu symetrické matice a determinantu matice) a vytvoření online testů, které by splňovali část testovací.

Pomocné nástroje pro procvičení byly navrhnuty tak, aby se uživateli v okně internetového prohlížeče zobrazil program, ve kterém bude možno práci s jednotlivými kapitolami vyzkoušet na příkladě, který si sám nadefinuje. Uživateli je umožněno využitím elementrárních operací upravovat matice do potřebných tvarů, získávat řešení soustav lineárních rovnic či počítat determinant matice. Jako nejvhodnější prostředek pro vytvoření těchto programů byl zvolen Java Applet.

Online testy z lineární algebry nabízejí spoustu příkladů k procvičení znalostí uživatele. Webová aplikace zobrazí jednotlivé příklady a čeká na uživatelovo řešení. Toto řešení je posléze načteno a na další stránce vyhodnoceno (správnost označených odpovědí a jejich počet). Jelikož přístup k příkladům uložených v XML dokumentech je zcela náhodný, je každý vygenerovaný test svým způsobem jedinečný. Protože bylo potřeba naprogramovat část, kdy se budou načítat data a poté bude prováděna práce s těmito daty, zvolil jsem jako nejlepší prostředek technologii pro tvorbu webových aplikací a služeb ASP.NET. Opět z důvodů, že jej lze provozovat na různých operačních systémech, bohatému výběru ovládacích prvků, knihoven tříd a šablon, a díky kompilovanému kódu také rychlosti aplikace. Pomocníkem pro výměnu dat mezi aplikacemi byl využit značkovací jazyk XML, který pomocí XML značek a obsahu mezi těmito značkami skvěle posloužil pro účely vytvoření testu a jeho zobrazení. Jelikož se tento test snaží uživatele prověřit ze znalostí lineární algebry, a jako celá matematika i tato používá matematické výrazy, vzorce a rovnice, bylo záhodno, aby výsledná stránka dokázala tyto vzorce správně graficky zobrazit. K těmto účelům byl využit jazyk MathML, podmnožina jazyka XML, kterou je možno připojit k HTML a interpretovat prohlížeči.

2 Java Applety

Požadavek byl takový, aby se uživateli v okně internetového prohlížeče zobrazil program, ve kterém bude možno práci s maticemi vyzkoušet a aplikovat tak své znalosti na příkladě, který si sám nadefinuje. Jako nejvhodnější prostředek byl zvolen Applet, tedy program vytvořený v programovacím jazyce Java. K tomuto rozhodnutí došlo na základě mé předchozí znalosti programovacího jazyka Java a se zkušenostmi s tvořením jednoduchých grafických Appletů. Dále vzhledem k možnosti jednoduchého zabudování Appletu do HTML stránky a tedy i prezentaci vytvořeného programu v okně internetového prohlížeče. V prospěch zvolení programovacího jazyka Java je i jeho silná vlastnost, kterou je platformová nezávislost, tedy v případě přítomnosti příslušného virtuálního stroje na dané platformě je spustitelný v libovolném operačním systému.

2.1 Applet

2.1.1 Popis appletu

Applet je při každé návštěvě webové stránky, v níž je obsažen, natažen do internetového prohlížeče a je spouštěn v prostředí Virtual Java Machine, které se na počítač instaluje spolu s prohlížečem. U některých prohlížečů (například MSIE) není instalace Virtual Java Machine (terminologie Microsoftu uvádí Microsoft Virtual Machine) obsažena v typické instalaci.

Applet pro svůj běh vyžaduje Java-kompatibilní prohlížeč. Nespouští se přímo (jako aplikace), nýbrž otevřením HTML dokumentu, kde je na něj umístěn odkaz pomocí speciální značky <*APPLET*> (viz kapitola 4.1).

Z bezpečnostních důvodů platí pro applet některá omezení a naopak má applet některé funkce rozšířeny:

- nemůže nahrávat knihovny ani definovat *nativní* metody.
- nemůže navazovat síťové spojení na jiný než domovský server.
- nemůže zapisovat do souborů na straně klienta (prohlížeče).
- nemůže spouštět programy na domovském serveru.
- nemůže číst některé systémové proměnné.
- může přehrávat zvuky
- Applet může požádat browser o zobrazení libovolné WWW stránky
- Applet může volat veřejné metody appletů umístěných na téže WWW stránce

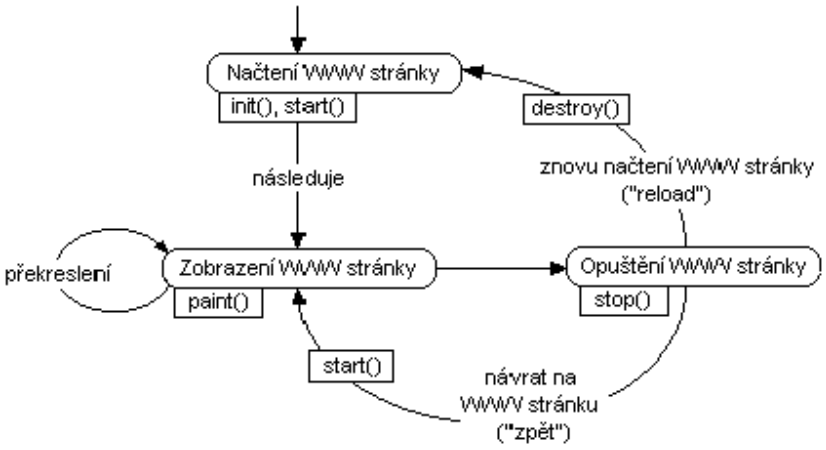
Základní strukturu appletu tvoří třída *java.applet.Applet*, která definuje základní metody tvořící rozhraní mezi prohlížečem a appletem. Program, který má fungovat jako applet, musí být potomkem této třídy.

Applet běží v grafickém kontextu, a jako takový je úzce spojen s "okenní" knihovnou *AWT*. Třída *java.applet.Applet* je potomkem třídy *java.awt.Panel*, která umožňuje appletu vlastnit komponenty uživatelského rozhraní, provádět grafický výstup a zachycovat události z klávesnice a myši.

Životní cyklus appletu závisí na prohlížeči, který během své činnosti volá tyto metody appletu:

- public void init() - při inicializaci,
- public void start() - při spuštění,
- public void paint(java.awt.Graphics g) - při překreslování,
- public void stop() - při zastavení,
- public void destroy() - při ukončení.

Tyto metody jsou ve třídě *java.applet.Applet* definovány jako prázdné a pro smysluplnou činnost je třeba v potomkovi překrýt alespoň jednu z nich.



znázorňení volání metod appletu v závislosti na stavu prohlížeče

2.1.2 Základní struktura appletů vytvořených v rámci této práce

Naprogramovány jsou applety pro čtvercové matice o rozměrech n řádků a n sloupců ($n \times n$). Přesto jsou k využití poskytnuty pouze ve čtyřech rozměrových variantách – matice 2×2 (o dvou řádcích a dvou sloupcích), $3 \times 3, 4 \times 4$ a 5×5. Je to hlavně ze dvou důvodů. Tím hlavním důvodem je velikost oblasti, kterou zabírají jednotlivé políčka pro vyplnění hodnoty daných prvků matice na plátně appletu. Při rozměrech matice 5×5 už applet zabírá i ve velikém rozlišení celou plochu obrazovky a při větších rozměrech už nejde applet načíst tak, aby byl graficky znázornitelný jak bylo plánováno. Navíc je pro výukové účely matice o větších rozměrech poměrně zbytečná.

Nejdříve byl implementován jako reprezentant hodnot jednotlivých prvků v maticích datový typ *double*. Postupem času však začalo vyvstávat několik problémů.

- Jedním z problémů, a v podstatě tím nejhlavnějším, byla zaokrouhlovací chyba při operacích s maticemi. Algoritmus například v LU rozkladu zachytává, zda-li náhodou uživatel již nenašel matici L. K tomu dochází ve chvíli, kdy je levá strana matice jednotková. Mohla však nastat situace, kdy se na diagonále objevilo číslo velice blízké jedničce (například 1,000000004) a to přesto, že uživatel žádnou chybu neudělal a vše bylo provedeno správně. Důsledkem je, že algoritmus toto nepojme jako čistou jedničku a v této chvíli už nemůže pokročit do další fáze výpočtu. Je to dáno tím, že při uložení čísel datového typu *double* dojde k určitému zaokrouhlení (dochází k zaokrouhlovacím chybám při převodu mezi desítkovým a dvojkovým vyjádřením reálného čísla v plovoucí čárce).

- Další věcí, která byla v tomto případě nutná, byla implementace metod pro dosažení výsledku. Mohla opět nastat situace popisovaná výše, kdy potřebujeme matici upravit na jednotkovou, aby proces výpočtu mohl přejít do další fáze. V případě, že by se ale na diagonále objevilo číslo větší než nula (například 7), tak už by nebylo možné docílit jednotkové matice za pomoci klasických elementárních operací (situaci by vyřešila například implementace metody pro dělení řádku matice).

Celkově pro větší efektivnost, přehlednost a intuitivní práci s maticemi byla navržena a implementována třída zlomek.

Třída zlomek má ve své podstatě při vytváření vždy dva parametry – čitatel a jmenovatel. V případě, že se jedná o celé číslo, pak je hodnotě prvku automaticky nastaven jmenovatel s hodnotou "1" (pokud se bude jednat o hodnotu "0" tak je i jmenovatel nastaven na hodnotu "0").

V algoritmu je systém převedení na zlomek vyřešen tak, že z textových polí se načte hodnota, která byla uživatelem zadána (ať už v celočíselném nebo racionálním tvaru). Je rozpoznáno lomítko, které rozděluje zlomek na dvě části a podle toho jsou hodnoty uloženy do jednoho nebo druhého parametru. Při výpisu matice do výstupní textové oblasti je vždy hodnota před zobrazením převedena do znakového řetězce pomocí metody *toString()* a poté zkontrolována. Pokud je její jmenovatel roven jedničce (nebo v případě nuly roven nule), je zobrazení celočíselné bez zlomkové čáry a jmenovatele. Výstup je také ošetřen metodou, která redukuje čitatel a jmenovatel (je-li možnost vykrátit zlomek a možnost dostat ho do základního tvaru). Další z možností této třídy je práce se znaménky. Metoda si projde zlomek a v případě záporných hodnot obou jeho parametrů převede tento zlomek na kladný. Dále je vždy snaha mít zápornou hodnotu v čitateli. Proto v případě záporného jmenovatele a kladného čitatele, je znaménko přesunuto do prvního parametru.

Vyjímkou je pouze řešení soustav lineárních rovnic zpětnou subtitucí, kde jsou hodnoty vždy zobrazovány v racionálním tvaru.

Problém se zaokrouhlováním a správnost zvolení implemetace zlomků je možno ukázat na příkladě (Příloha 7.1).

Při využití zlomků jako objektů bylo potřeba dořešit způsob kopírování či přiřazování jinému objektu. Objekt totiž zkopíruje ukazatel na místo v paměti místo toho, aby zkopíroval přímo hodnotu. V tomto případě došlo například k tomu, že pomocí dvou cyklů bylo potřeba zkopírovat jednu matici a její jednotlivé prvky vložit do druhé. Poté při práci s nově vytvořenou maticí nedocházelo ke změnám hodnot, protože se na místa prvků v matici nezkopírovaly hodnoty, ale pouze ukazele na místo v paměti, které obsahovaly hodnoty předešlé matice. Proto byla potřeba implementovat metodu, která zkopírovala přímo hodnoty jednotlivých prvků z jedné matice do druhé matice, a to tak, že daným prvkům přímo zkopírovala zvlášť čitatel a zvlášť jmenovatel.

Dalším nedostatkem práce s třídou zlomek je, že nejde s nimi pracovat jako s normálními čísly. Znamená to, že každou operaci (sčítání, odčítání, násobení, dělení), kterou s nimi je potřeba provádět, bylo také potřeba kompletně naprogramovat. To v sobě schovávalo mimo jiné například vytvoření funkcí jako hledání největšího společného dělitele, nejmenšího společného násobku či redukce zlomku. Proto byl také pro snadnější práci s touto třídou implementován jednoduchý přístup k jeho parametrům pomocí metod *set()* a *get()*.

Aby zobrazované matice vypadaly i graficky jako matice, bylo využito speciálních ASCII znaků (uveden je ASCII kód znaku a jeho grafický výstup).

250C		2510
2502	[ ]	2502
2514		2518

Struktura appletů

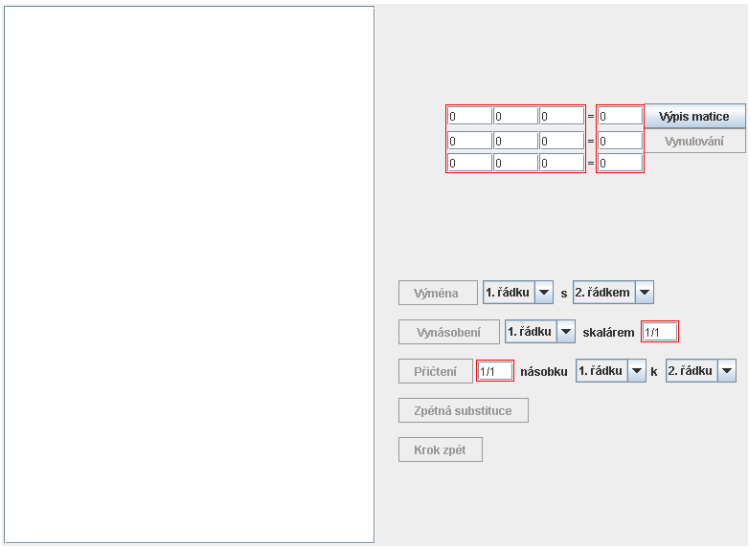
JTextArea

Jednou z komponent, která je společná všem appletům, je (JTextArea). Je to multi-řádková oblast, která zobrazuje jednoduchý text. Všechno co uživatel provádí má za následek výstup do této textové oblasti. Nevýhodou této komponenty je, že jakmile chce zobrazit jakýkoliv znak či řetězec znaků, tak toto místo pro daný počet znaků vytvoří, poté zobrazí a pokračuje dál. To má za následek nemožnost jakéhokoliv posunu zpět o určitý počet znaků a nemožnost posunu na předchozí řádek. Není tedy možno formátovat již zobrazený text a není možno přídávat znaky na pozice v textové oblasti, které již byly zaplněny. Z tohoto důvodu nebylo možno dostatečně zpřehlednit a graficky zdokonalit zobrazování matic (například pokud počet znaků na jednom řádku před levou hranatou závorkou uzavírající matici byl jiný než na dalším řádku, pak závorka není až tak graficky dokonalou závorkou).

JTextArea v appletu

JTextField

Textová políčka (JTextField) umožňují uživateli zadat matici, na kterou je možno poté aplikovat elementární úpravy a dojít tak až ke konečnému výsledku. Jak už bylo zmíněno, hodnoty matice je možno zadávat jak ve tvaru celočíselném, tak ve tvaru racionálním pomocí znaku "/"(tedy například "$3/5$"). Stejným způsobem lze zadávat hodnoty skalárů v rámci elementárních operací "vynásobení řádku" a "sčítání dvou řádků". Na počátku spuštění appletu je do polí určujících vstupní matici automaticky vložena hodnota „0“ (zadávat lze do těchto polí jakékoliv znaky, budou-li se však odlišovat od celočíselného nebo racionálního tvaru čísla, je jako hodnota namísto znaku uložena hodnota „0“). Oproti tomu do textových polí pomáhajících u elementárních operací pro vynásobení řádku a pro sčítání dvou řádků jsou automaticky nastaveny hodnoty zlomku „$1/1$“ (v obou případech nelze použít hodnotu „0“, která je ošetřena vyjímkou – ve výstupní textové oblasti se zobrazí, že nelze násobit tímto zlomkem).



JTextField v appletu

JComboBox

Při využívání elementárních operací je často potřeba vybírat různé indexy řádků (popřípadě sloupců), na které má být daná operace aplikována. K těmto účelům je zde komponenta JComboBox (můžeme ji chápat jako nějakou rozbalovací se nabídku možností). Po spuštění je automaticky nastaven tento box na první index v nabídce (pokud jsou pro danou operaci v možnostech dva JComboBoxy, pak je první nastaven na první možnou hodnotu v nabídce a druhý na druhou možnost).

JComboBox v appletu

JButton

Další komponentou využitou při vytváření appletů je tlačítko (JButton). Komponenta, která po jejím stisknutí vykoná operace, které jsou pod tímto tlačítkem implementovány.

- Výpis matice – vypíše matici, která byla zadána uživatelem a následně, pokud úspěšně splní podmínky určené k dalším krokům, vypíše další části procesu (až do chvíle, kdy některou z podmínek nesplní, nebo do chvíle, kdy je výpočet úspěšně dokončen).

- Vynulování – Vymaže obsah výstupní textové oblasti a znovu do ní vloží úvodní text. Nastaví hodnoty ve všech textových polích určujících vstupní matici na hodnotu „0“. Textová pole pomáhající u elementárních operací pro vynásobení řádku a pro sčítání dvou řádků jsou nastavena na hodnotu zlomku rovnu „$1/1$“. Všechny matice, které jsou důležité u výpočtu, jsou znovu inicializovány a všem hodnotám těchto matic je přiřazena hodnota „0“. Indexy každého z JComboBoxů nastaví na původní hodnoty. Nakonec jsou všechna ostatní tlačítka (kromě tlačítka „Výpis“) zakázána.

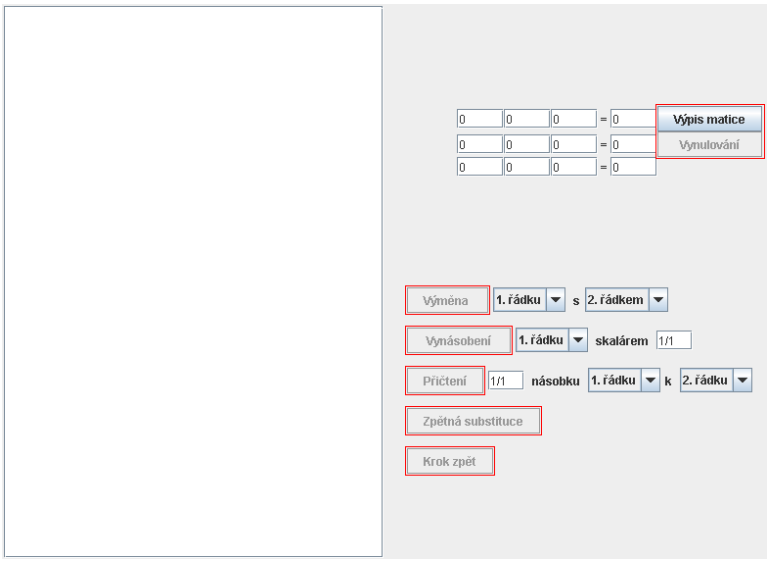
- Výměna – vymění řádek (popřípadě sloupec), jehož index je zadán v prvním JComboBoxu, s řádkem (popřípadě sloupcem), jehož index je zadán ve druhém JComboBoxu.

- Vynásobení – vynásobí řádek (s indexem uvedeném v JComboBoxu) skalárem, který je zadán uživatelem v příslušném textovém poli.

- Přičtení – přičte násobek (definovaný odpovídajícím textovým polem) řádku, který je indexem uveden v prvním JComboBoxu, k řádku s příslušným indexem uvedeném v druhém JComboBoxu.

- Krok zpět – provede kompletní návrat o jednu provedenou operaci zpět. Liší se pouze v případě LU rozkladu, kde nelze používat krok zpět ve chvíli, kdy se proces LU rozkladu přesune do další fáze výpočtu. Lze tedy krok zpět využít vždy jen v rámci jedné fáze výpočtu (nelze například použít krok zpět ihned poté co získáme horní trojúhelníkovou matici a přejdeme do fáze hledání dolní trojúhelníkové matice. Lze ale krok zpět využít ve chvíli, kdy stále hledáme dolní trojúhelníkovou matici).

- Zpětná substituce – je implementováno pouze u appletu s Gaussovou eliminací. Jakmile je dosaženo horní trojúhelníkové matice, je výpočet u konce a je aktivováno toto tlačítko pro získání řešení soustav lineárních rovnic.



JButton v appletu

2.2 Teorie shrnující oblast matic

Zabýváme se řešením obecných soustav lineárních rovnic, jejichž cílem je najít pro jednotlivé úlohy řešení $x_1,\dots,x_n$ tak, aby pro daná čísla a_{ij} a b_i, $i=1,\dots,m$, $j=1,\dots,n$ platilo

$$\begin{array}{ccccccc} a_{11}x_1 & + & \dots & + & a_{1n}x_n & = & b_1 \\ \vdots & & \dots & & \vdots & & \vdots \\ a_{m1}x_1 & + & \cdots & + & a_{mn}x_n & = & b_m \end{array}. \qquad (1)$$

Součástí této úlohy je rozhodnout, zda vůbec nějaké řešení dané soustavy existuje, kolik jich je a co o nich lze říci v případě, že je jich nekonečně mnoho.

Ekvivalentní úpravy

Základní myšlenkou řešení soustavy lineárních rovnic je nahradit danou soustavu jinou soustavou, která bude mít stejné řešení a bude jednodušší. Například v případě dvou rovnic se dvěma neznámými je jednodušší taková soustava, která obsahuje alespoň jednu rovnici s pouze jednou neznámou. Tuto rovnici poté už můžeme řešit nezávisle na druhé rovnici. Novou soustavu můžeme dostat používáním tzv. Ekvivalentních úprav (řešení původní soustavy je i řešením soustavy upravené):

- (E1) Vzájemná výměna libovolných dvou rovnic soustavy
- (E2) Násobení obou stran některé rovnice soustavy nenulovým číslem
- (E3) Příčtení násobku některé rovnice soustavy k jiné rovnici

Maticový zápis

Během úprav těchto rovnic si můžeme značně ušetřit práci, když nebudeme opisovat neznámé, ale když budeme soustavu rovnic v tomto úsporném režimu zapisovat do tabulky

$$\left[\begin{array}{ccc|c} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} & b_m \end{array}\right], \qquad (2)$$

kterou nazýváme rozšířená matice soustavy. Maticí soustavy nazýváme tu část tabulky bez posledního sloupce. Pravou stranou soustavy zase nazýváme poslední sloupec tabulky.

Ekvivalentním úpravám soustavy rovnic odpovídají operace s řádky rozšířené matice soustavy, které nazýváme elementární (řádkové) operace:

- (e1) Vzájemná výměna libovolných dvou řádků
- (e2) Násobení některého řádku nenulovým číslem
- (e3) Přičtení násobku některého řádku k jinému řádku

Úprava na schodový tvar

Pomocí elementárních řádkových operací můžeme převést matici (2) na tzv. schodový tvar, to znamená na tvar, v němž jsou první nenulové prvky řádků zvané vedoucí prvky uspořádány jako schody klesající z levé strany na pravou. Je ale požadováno, aby vedoucí prvky nebyly nad sebou a aby všechny případné nulové řádky byly ty nejspodnější.

Poté při úpravě matic využijeme pozorování, že je-li v matici (2) prvek a_{ij} nenulový, pak jestliže vynásobíme i-tý řádek této matice číslem $-a_{kj}/a_{ij}$ a přičteme-li ho ke k-tému řádku, bude mít upravená matice v k-tém řádku a j-tém sloupci prvek

$$a_{kj} + \left(-a_{kj}/a_{ij}\right)a_{ij} = 0\,.$$

Pokud je prvek a_{11} nenulový, lze takto transformovat matici (1) na tvar

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22}^1 & \dots & a_{2n}^1 & b_2^1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{m2}^1 & \dots & a_{mn}^1 & b_m^1 \end{array}\right].$$

V případě, že bude také prvek a_{22}^1 nenulový, můžeme podobně dosáhnout pomocí elementárních řádkových operací toho, že i pod tímto prvkem budou v upravené matici nuly. Bude-li vždy $a_{ii}^{i-1} \neq 0$, dostaneme nakonec matici ve schodovém tvaru s nenulovými prvky $a_{11}, a_{22}^1, \dots, a_{kk}^{k-1}$.

$$\left[\begin{array}{ccccccc|c} a_{11} & a_{12} & \dots & a_{1k} & \dots & a_{1n} & b_1 \\ 0 & a_{22}^1 & \dots & a_{2k}^1 & \dots & a_{2n}^1 & b_2^1 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & 0 & & a_{kk}^{k-1} & \dots & a_{kn}^{k-1} & b_k^{k-1} \\ 0 & 0 & \dots & 0 & \dots & 0 & b_{k+1}^k \\ 0 & 0 & \dots & 0 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 & 0 \end{array}\right]. \qquad (3)$$

Nenulové prvky v levé části upravené matice soustavy připomínají trojúhelník, proto říkáme, že matice je v trojúhelníkovém tvaru. Úpravu na trojúhelníkový tvar můžeme provést i v případě, že pokaždé když $a_{ii}^{i-1} = 0$, je možno nalézt prvek $a_{ji}^{i-1} \neq 0, j > i$. Stačí vzájemně vyměnit před úpravou i-tý a j-tý řádek.

Zpětná substituce

Zpětná substituce je získání řešení soustavy z matice ve schodovém tvaru. Pro naše účely ji budeme potřebovat při výpočtu řešení soustav lineárních rovnic pomocí Gaussovy eliminace či LU rozkladu. Rozlišujeme tři případy, které mohou nastat:

- V případě, že poslední nenulový řádek rozšířené matice soustavy má nenulový pouze poslední prvek b_{k+1}^{k}, pak tomuto řádku odpovídá rovnice

  $$0 = b_{k+1}^{k},$$

  která nemá pro $b_{k+1}^{k} \neq 0$ řešení. V tomto případě tedy daná soustava nemá řešení.

- V případě, že rozšířená matice má trojúhelníkový tvar (3) s $k = n, b_{n+1}^{n} = 0$ a $a_{ii}^{i-1} \neq 0$, $i = 1, \ldots, n$, pak n-tá rovnice má tvar

  $$a_{nn}^{n-1} x_n = b_n^{n-1},$$

  Ze které snadno vyčteme x_n. Po dosazení do předchozích rovnic zůstane v $(n-1)$-ní rovnici pouze jediná neznámá, kterou můžeme snadno vypočíst. Budeme-li takto postupovat dále, určíme snadno jediné řešení soustavy.

- V případě, že rozšířená matice má obecný schodový tvar, pak z každé rovnice soustavy vyjádříme neznámou, která odpovídá vedoucímu prvku. Postupným dosazováním od posledního řádku dostaneme vzorce pro neznámé odpovídající vedoucím prvkům vyjádřené pomocí neznámých na pravé straně. V tomto případě má soustava nekonečně mnoho řešení.

Permutační matice

Matice P se nazývá permutačni matice, je-li možno P získat z jednotkové matice I stejného typu postupnou výměnou sloupců.

Například $$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} s_1 \leftrightarrow s_2 \sim \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} s_1 \leftrightarrow s_3 \sim \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = P.$$

Symetrická matice

Pokud je transponovaná matice shodná s původní maticí, tzn. $A^T = A$, pak matici A označujeme jako symetrickou. Pro prvky symetrické matice platí

$$a_{ij} = a_{ji}.$$

Trojúhelníkové matice

Čtvercová matice L se nazývá dolní trojúhelníková matice, jestliže má nad diagonálou všechny prvky nulové. Pro prvky l_{ij} dané dolní trojúhelníkové matice L tedy platí $l_{ij} = 0$ pro $i < j$.

Čtvercová matice U se nazývá horní trojúhelníková matice, jestliže má pod diagonálou všechny prvky nulové. Pro prvky u_{ij} dané horní trojúhelníkové matice U tedy platí $u_{ij} = 0$ pro $i > j$.

Matice L je tedy dolní trojúhelníková, právě když L^T je horní trojúhelníková matice.

Diagonální redukce pozitivně definitní matice

Při řešení kongruencí jsou využívány elementární řádkové operace a jejich maticový zápis. Novinkou však je, že ke každé elementární operaci následně uvažujeme i její sloupcovou variantu a místo o elementárních operacích je možno mluvit o elementárních kongruencích. Je-li tedy T matice některé elementární operace s řádky čtvercové matice A, pak matici upravenou příslušnou elementární kongruencí lze zapsat ve tvaru TAT^T.

Při spojení tohoto pozorování s postupy, které jsou spojeny s LU rozkladem, snadno je možno dokázat následující tvrzení, které lze použít k ověření pozitivní definitnosti matice nebo k řešení soustav s pozitivně definitní maticí.

Pokud je A je pozitivně definitní matice. Pak existuje regulární dolní trojúhelníková matice L a diagonální matice D s kladnou diagonálou tak, že

$$LAL^T = D.$$

Kongruentní matice

Čtvercová matice A je kongruentní s maticí B, jestliže existuje regulární matice T tak, že

$$A = T^T BT$$

Symetrická matice kongruentní s diagonální maticí je pozitivně definitní, právě když tato diagonální matice má kladné diagonální prvky.

2.3 Applety vybraných kapitol lineární algebry

2.3.1 Gaussova eliminace

Gaussova eliminační metoda (Gaussova eliminace) je metodou exaktního řešení soustavy lineárních algebraických rovnic. Obecně řečeno, Gaussova eliminace představuje řešení problému, který je vyjádřen pomocí matice, převedením dané matice na horní trojúhelníkovou matici nebo na diagonální matici.

Výpočetní postup pro řešení soustav lineárních rovnic se nazývá Gaussova eliminace. Tento postup je rozdělen do dvou kroků. Prvním krokem je redukce na schodový tvar, která se při řešení soustav nazývá dopředná redukce. Druhým krokem je výpočet řešení soustavy se schodovou maticí - zpětná substituce.

Algoritmus Gaussovy eliminace

/* Hodnoty matice jsou uloženy v dvojrozměrném poli *matrix*. Matice je v tomto algoritmu o rozměrech $N \times N$. Řešení zpětné substituce je ukládáno do vektoru *vysledek* */

```
/* inicializace proměnné, která má hodnotu odpovídající rozměru čtvercové matice */
int N = 4;

/* algoritmus dopředné redukce */
public void gaussovaEliminace()
{
    /* inicializace proměnných pro průchod maticemi */
    int i, j, k;

    /* inicializace proměnné, do které se uloží index řádku s největší hodnotou prvku v daném
       sloupci */
    int max;

    /* inicializace pomocné proměnné pro potřeby výměny řádků
    double t;

    for (i = 0; i < N; i++)
    {
        max = i;
        for (j = i + 1; j < N; j++)

            /* jestliže bude hodnota v daném sloupci vetší na dalším řádku, ulož index tohoto
            řádku */
            if (matrix[j][i] > matrix[max][i])
            max = j;
```

```
                /* cyklus, ve kterém dojde k výměně řádků (pokud je potřeba) */
                for (j = 0; j < N + 1; j++)
                {
                        t = matrix[max][j];
                        matrix[max][j] = matrix[i][j];
                        matrix[i][j] = t;
                }

                /* cyklus, ve kterém dochází k samotné Gaussově eliminaci */
                for (j = N; j >= i; j--)
                {
                        for (k = i + 1; k < N; ++k)
                        {
                                matrix[k][j] -= matrix[k][i] / matrix[i][i] * matrix[i][j];
                        }
                }
        }
}

/* algoritmus zpětné substituce */
public void zpetnaSubstituce()
{
        for (int j = N - 1; j >= 0; j--)
        {
                /* inicializece proměnné, do které se ukládají vypočítáváné hodnoty */
                double sum = 0.0;

                for (int k = j + 1; k < N; k++)

                /* výpočet pomocné proměnné z již vypočítaných řešení */
                sum += matice[j][k] * vysledek[k];

                /* uložení jednotlivých řešení do vektoru vysledek, přičemž hodnota řešení je dosažena
                   jako odečtení již získaných řešení od pravé strany soustavy a poté vydělení tohoto
                   čísla prvkem diagonálním */
                vysledek[j] = (matice[j][N] - sum) / matice[j][j];
        }
}
```

Applet Gaussovy eliminace

Applet umožňuje uživateli pomocí elementárních operací provést kompletní Gaussovu eliminaci. Je rozložen do dvou částí. V první části (poté co uživatel zadá příslušnou matici) je cílem upravit matici na horní trojúhelníkovou (tedy provést dopřednou redukci). K tomuto je možné využít elementární (řádkové) operace. Jakmile se podaří horní trojúhelníkovou matici získat, je na řadě už jen jeden jediný krok a tím je zpětná substituce. Z ní už je poté výstupem do textové oblasti výsledek ve formě řešení soustav lineárních rovnic (parametrické řešení soustavy lineárních rovnic není podporováno).

Příklad:

$$\begin{aligned} 3x_1 - 2x_2 + 2x_3 &= 10 \\ x_1 + 3x_2 - x_3 &= 2 \\ 2x_1 + 2x_2 + 3x_3 &= 15 \end{aligned}$$

Soustavu si přepíšeme:

$$A = \begin{bmatrix} 3 & -2 & 2 \\ 1 & 3 & -1 \\ 2 & 2 & 3 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, b = \begin{bmatrix} 10 \\ 2 \\ 15 \end{bmatrix}$$

Gaussovou eliminací upravíme rozšířenou matici soustavy:

$$\left[\begin{array}{ccc|c} 3 & -2 & 2 & 10 \\ 1 & 3 & -1 & 2 \\ 2 & 2 & 3 & 15 \end{array}\right] \cdot 3 \sim \left[\begin{array}{ccc|c} 3 & -2 & 2 & 10 \\ 3 & 9 & -3 & 6 \\ 6 & 6 & 9 & 45 \end{array}\right] \begin{matrix} \\ -r_1 \\ -2r_1 \end{matrix} \sim \left[\begin{array}{ccc|c} 3 & -2 & 2 & 10 \\ 0 & 11 & -5 & -4 \\ 0 & 10 & 5 & 25 \end{array}\right] \cdot \frac{1}{5} \sim \left[\begin{array}{ccc|c} 3 & -2 & 2 & 10 \\ 0 & 11 & -5 & -4 \\ 0 & 2 & 1 & 5 \end{array}\right] \cdot 11 \sim$$

$$\sim \left[\begin{array}{ccc|c} 3 & -2 & 2 & 10 \\ 0 & 11 & -5 & -4 \\ 0 & 22 & 11 & 55 \end{array}\right] \begin{matrix} \\ \\ -2r_2 \end{matrix} \sim \left[\begin{array}{ccc|c} 3 & -2 & 2 & 10 \\ 0 & 11 & -5 & -4 \\ 0 & 0 & 21 & 63 \end{array}\right] \cdot \frac{1}{21} \sim \left[\begin{array}{ccc|c} 3 & -2 & 2 & 10 \\ 0 & 11 & -5 & -4 \\ 0 & 0 & 1 & 3 \end{array}\right]$$

$$\begin{aligned} 3x_1 - 2x_2 + 2x_3 &= 10 \\ 11x_2 - 5x_3 &= -4 \\ x_3 &= 3 \end{aligned}$$

Dosazením poslední rovnice do druhé dostaneme

$$11x_2 - 5 \cdot 3 = -4 \Leftrightarrow x_2 = 1$$

A následně dosazením do první rovnice dostaneme

$$3x_1 - 2 \cdot 1 + 2 \cdot 3 = 10 \Leftrightarrow x_1 = 2$$

Řešením je tedy vektor $x = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$

Nyní řešení stejného problému pomocí appletu:

1. Do textových polí nejdříve zadáme matici, kterou chceme počítat a poté zmáčkneme tlačítko *Výpis matice*.
2. Druhý a třetí řádek vynásobíme hodnotou „3“ tak, že do příslušného políčka operace *Vynásobení* vyplníme hodnotu „3“ a poté zvolíme postupně indexy druhého a třetího řádku.
3. Dalším krokem je odečtení prvního řádku od druhého řádku. To provedeme vyplněním hodnoty „-1“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého na index druhého řádku.
4. Dále je odečtení dvojnásobku prvního řádku od třetího řádku řádku. To provedeme vyplněním hodnoty „-2“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého na index třetího řádku.
5. Třetí řádek vynásobíme hodnotou „$1/5$“ tak, že do příslušného políčka operace *Vynásobení* vyplníme hodnotu „$1/5$“ a poté zvolíme index třetího řádku.
6. Třetí řádek vynásobíme hodnotou „11“ tak, že do příslušného políčka operace *Vynásobení* vyplníme hodnotu „11“ a poté zvolíme index třetího řádku.
7. Dále je odečtení dvojnásobku druhého řádku od třetího řádku. To provedeme vyplněním hodnoty „-2“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého JComboBoxu na index třetího řádku.
8. Třetí řádek vynásobíme hodnotou „$1/21$“ tak, že do příslušného políčka operace *Vynásobení* vyplníme hodnotu „$1/21$“ a poté zvolíme index třetího řádku.
9. Matice je upravena na schodový tvar. Zbytek operaci za nás provede algoritmus pro výpočet zpětné substituce. Stačí tedy na závěr stisknout tlačítko *Zpětná substituce*.

2.3.2 Gauss-Jordanova eliminační metoda

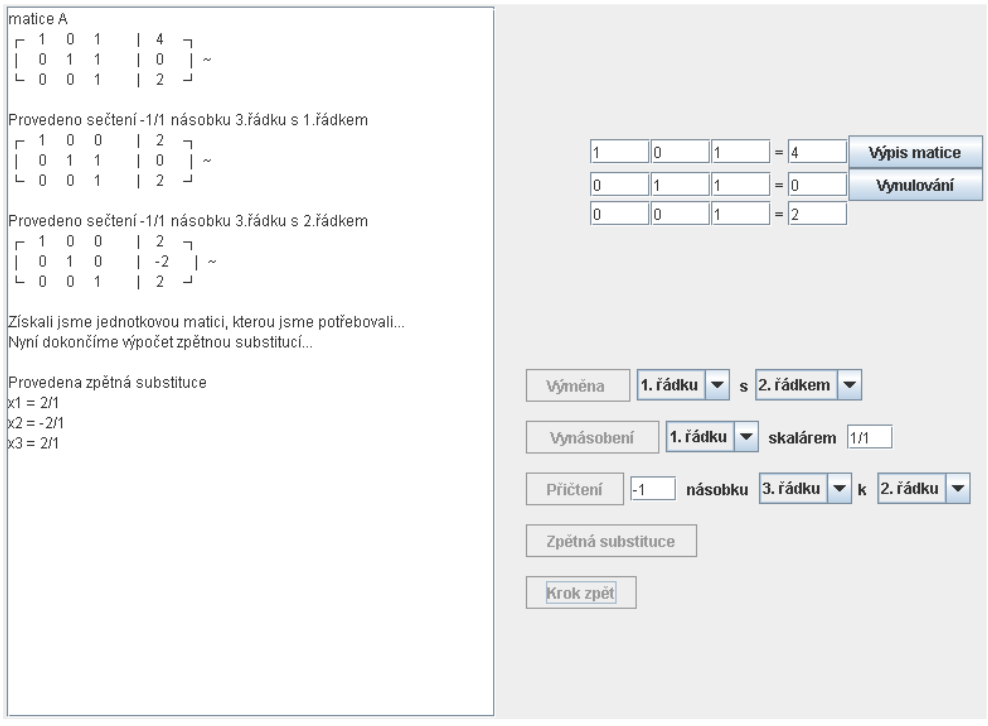
Cílem Gaussovy-Jordanovy eliminační metody je získat řešení soustav lineárních rovnic. Je nutno pomocí elementárních operací převést rozšířenou matici soustavy na matici, kde na místě původní matice soustavy bude matice v normovaném tvaru a na místě pravých stran se pak vyskytne řešení soustavy.

Budeme říkat, že matice je v normovaném schodovém tvaru, jestliže je v takovém schodovém tvaru, že všechny prvky nad vedoucími prvky jsou nulové a navíc vedoucí prvky jsou rovny jedné.

Gauss-Jordanova metoda se od Gaussovy eliminace liší tím, že se při dopředné redukci upraví rozšířená matice soustavy na normovaný schodový tvar místo na schodový tvar. Zpětná substituce je pak snadnější a nemusí se provádět od poslední rovnice.

Applet Gauss-Jordanovy eliminační metody

Applet je totožný s appletem klasické Gaussovy eliminace. Pouze je zde změna v tom, že pokračuje v upravování matice až do tvaru jednotkové matice. Řešení soustav lineárních rovnic je pak dokončeno opět pomocí tlačítka *Zpětná substituce*, které je zobrazeno do výstupní textové oblasti.



Příklad práce s appletem Gauss-Jordanovy eliminace

2.3.3 LU rozklad

Cílem LU rozkladu je rozložit původní matici na matice trojúhelníkové a s jejich pomocí získat například řešení soustav lineárních rovnic (či nalezení inverzní matice).

Věta: Pokud A je regulární čtvercová matice, pak existuje dolní trojúhelníková matice L, horní trojúhelníková matice U a permutační matice P tak, že

$$AP = LU$$

Matice L, U jsou regulární.

Řešení soustav pomocí LU rozkladu

Řešení soustav pomocí trojúhelníkového rozkladu spočívá v postupném řešení soustav s maticemi L, U a $\tilde{P}$. Jestliže tedy $A = LU\tilde{P}$, pak místo soustavy $Ax = b$ budeme řešit soustavu

$$L\left(U\left(\tilde{P}x\right)\right) = b$$

tak, že postupně vyřešíme

$$Lz = b, \quad Uy = z, \quad \tilde{P}x = y$$

Vypočet LU rozkladu

Postupným upravováním matice $\left[A \mid I\right]$ na tvar $\left[U \mid \tilde{L}\right]$ pomocí elementárních operací, avšak bez použití výměny řádků. Tím dosáhneme toho, že matice $\tilde{L}$ bude dolní trojúhelníková. Je-li to nutné, provádíme místo výměny řádků výměny sloupců, které neprovádíme jen na matici A, ale také zvlášť na další jednotkové matici, která se postupně transformuje na P. Dolní trojúhelníkovou matici L dostaneme inverzí matice $\tilde{L}$, tedy s pomocí elementárních řádkových operací, které převedou matici $\left[\tilde{L} \mid I\right]$ na $\left[I \mid L\right]$.

Applet LU rozkladu

Applet LU rozkladu je rozložen do několika fází. V první části (poté co uživatel zadá příslušnou matici) je cílem upravit matici na horní trojúhelníkovou matici (tedy provést dopřednou redukci matice, jejíž levá strana je matice, kterou jsme zadali a jejíž pravá strana je matice jednotková). Dopředná redukce znamená, že všechny prvky pod diagonálou matice musí být nulové. K tomuto je možné využít elementární (řádkové) operace, které mění jak pravou, tak levou stranu matice. Nesmíme ale zapomenout na to, že záměna sloupců má vliv na matici P, na kterou se tato výměna sloupců aplikuje taktéž. Jakmile se podaří horní trojúhelníkovou matici získat (dostali jsme tím matici U), můžeme pokračovat v další fázi a tou je hledání matice L. Matici L hledáme tak, že pravou stranu z první fáze při hledání matice U vezmeme a vložíme ji na levou stranu a jako pravou stranu opět

dosadíme matici jednotkovou. Matici řešíme tak dlouho, dokud není její levá strana jednotková (jakmile bude levá strana jednotková - nalezena matice L). V této chvíli se v textovém poli zobrazí zkouška, ve které by měla být splněna rovnost $AP = LU$.

Další fází je řešení soustav lineárních rovnic pomocí LU rozkladu. Řešíme postupně soustavy $Lz = b$, poté $Uy = z$ a nakonec získáváme řešení soustavy lineárních rovnic z $x = Py$. Přestože je na první pohled zřejmý výsledek, je zde stále nutno využít elementárních operací a upravit matici na horní trojúhelníkovou (algoritmus totiž nezná zpětnou substituci pro dolní trojúhelníkovou matici), aby byl výsledek ještě zřetelnější a algoritmus mohl pokračovat dál. Po získaném konečném řešení je opět v textové oblasti zobrazena zkouška, zda-li opravdu $Ax = b$.

Příklad:

$$\begin{aligned} 2x_2 - x_3 &= 1 \\ x_1 - x_2 + x_3 &= 0 \\ x_1 - 3x_2 + 4x_3 &= 0 \end{aligned}$$

Nalezení matice U:

$$\left[\begin{array}{ccc|ccc} 0 & 2 & -1 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 & 1 & 0 \\ 1 & -3 & 4 & 0 & 0 & 1 \end{array}\right] s_1 \Leftrightarrow s_3 \sim \left[\begin{array}{ccc|ccc} -1 & 2 & 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 & 1 & 0 \\ 4 & -3 & 1 & 0 & 0 & 1 \end{array}\right] +r_1 \sim$$

$$\sim \left[\begin{array}{ccc|ccc} -1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 5 & 1 & 4 & 0 & 1 \end{array}\right] +(-5)r_2 \sim \left[\begin{array}{ccc|ccc} -1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & -4 & -1 & -5 & 1 \end{array}\right],$$

$$U = \begin{bmatrix} -1 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & -4 \end{bmatrix},\ \tilde{L} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & -5 & 1 \end{bmatrix}, \qquad \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] s_1 \Leftrightarrow s_3 \sim \left[\begin{array}{ccc} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{array}\right] = P$$

Výpočet $\tilde{L}^{-1}$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ -1 & -5 & 1 & 0 & 0 & 1 \end{array}\right] -r_1 \sim \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 \\ 0 & -5 & 1 & 1 & 0 & 1 \end{array}\right] +5r_2 \sim \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -4 & 5 & 1 \end{array}\right] +5r_2$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -4 & 5 & 1 \end{bmatrix}$$

Řešení soustavy

$Lz = b$:

$z_1 = 1$

$$-z_1 + z_2 = 0 \quad \Rightarrow \quad -1 + z_2 = 0 \quad \Rightarrow \quad z_2 = 1 \quad \Rightarrow \quad z = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$$

$$-4z_1 + 5z_2 + z_3 = 0 \qquad -4 + 5z_2 + z_3 = 0 \qquad z_3 = -1$$

$Uy = z$:

$$-y_1 + 2y_2 = 1 \qquad -y_1 + 2y_2 = 1 \qquad y_1 = \frac{1}{2}$$

$$y_2 + y_3 = 1 \quad \Rightarrow \quad y_2 + y_3 = 1 \quad \Rightarrow \quad y_2 = \frac{3}{4} \quad \Rightarrow \quad y = \begin{pmatrix} \frac{1}{2} \\ \frac{3}{4} \\ \frac{1}{4} \end{pmatrix}$$

$$-4y_3 = -1 \qquad y_3 = \frac{1}{4}$$

$x = Py$:

$$x = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 1/2 \\ 3/4 \\ 1/4 \end{pmatrix} = \begin{pmatrix} 1/4 \\ 3/4 \\ 1/2 \end{pmatrix}$$

Řešením je $x_1 = 1/4$, $x_2 = 3/4$, $x_3 = 1/2$

Nyní řešení stejného problému pomocí appletu:

1. Do textových polí nejdříve zadáme matici, kterou chceme počítat a poté zmáčkneme tlačítko *Výpis matice*.
2. První a třetí sloupec vyměníme využitím první elementární operací v nabídce (*Výměna*) a zvolením prvního sloupce v prvním JComboBoxu a třetího sloupce v druhém JComboBoxu.
3. Dalším krokem je přičtení prvního řádku k druhému řádku. To provedeme vyplněním hodnoty „1“ v elementární operaci *Přičtení* a nastavením prvního **J**ComboBoxu na index prvního řádku a druhého JComboBoxu na index druhého řádku.
4. Dalším krokem je přičtení čtyřnásobku prvního řádku k třetímu řádku. To provedeme vyplněním hodnoty „4“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého JComboBoxu na index třetího řádku.
5. Dále je přičtení mínus pětinásobku druhého řádku k třetímu řádku. To provedeme vyplněním hodnoty „-5“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého JComboBoxu na index třetího řádku.
6. Provede se ukončení první fáze, kdy jsme nalezli matici U. Dále se snažíme dostat z matice $\tilde{L}$ jednotkovou matici.
7. Dalším krokem je odečtení prvního řádku od druhého řádku. To provedeme vyplněním hodnoty „-1“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého JComboBoxu na index druhého řádku.
8. Dalším krokem je přičtení prvního řádku k třetímu řádku. To provedeme vyplněním hodnoty „1“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého JComboBoxu na index třetího řádku.
9. Dále je přičtení pětinásobku druhého řádku k třetímu řádku. To provedeme vyplněním hodnoty „5“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého JComboBoxu na index třetího řádku.
10. Získali jsme matici L. Ve výstupní textové oblasti je vypsána zkouška, kde by se mělo rovnat $AP = LU$. Algoritmus skočí do další fáze. Přestože je na první pohled zřejmý výsledek, je zde stále nutno využít elementárních operací a upravit matici na horní trojúhelníkovou, aby byl výsledek ještě zřetelnější a algoritmus mohl pokračovat dál.
11. Dále je přičtení prvního řádku k druhému řádku. To provedeme vyplněním hodnoty „1“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého JComboBoxu na index druhého řádku.
12. Dalším krokem je přičtení čtyřnásobku prvního řádku k třetímu řádku. To provedeme vyplněním hodnoty „4“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého JComboBoxu na index třetího řádku.
13. Dále je přičtení mínus pětinásobku druhého řádku k třetímu řádku. To provedeme vyplněním hodnoty „5“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého JComboBoxu na index třetího řádku.
14. Získali jsme tím řešení *z*. V dalším kroku dostáváme hned řešení *y*. A v posledním kroce je vypočítáno výsledné řešení *x*. Všechny tyto kroky jsou vypsány do textové oblasti, kde se na závěr ještě objeví zkouška zda-li se $Ax = b$.

2.3.4 Inverzní matice

Inverzní matice k původní matici je taková matice, která splňuje vztahy $A^{-1}A = I = AA^{-1}$. Pokud je A čtvercová matice řádu n, pak rovnice $AX = I$ má jediné řešení X pravě tehdy, když A je regulární. V tom případě platí $X = A^{-1}$.

Inverzní matici získáme tedy úpravou matice, kde levou stranu tvoří námi zadaná matice, a pravou stranu tvoří matice jednotková. Aplikováním elementárních operací se snažíme na levé straně získat jednotkovou matici a tím na pravé straně získat matici, kterou hledáme – inverzní matici.

Příklad:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

Postupnou úpravou rozšířené matice pro soustavu $AX = I$ dostaneme

$$\left[A \mid I\right] = \left[\begin{array}{cc|cc} 2 & -1 & 1 & 0 \\ -1 & 2 & 0 & 1 \end{array}\right] r_2 + \tfrac{1}{2} r_1 \mapsto \left[\begin{array}{cc|cc} 2 & -1 & 1 & 0 \\ 0 & \tfrac{3}{2} & \tfrac{1}{2} & 1 \end{array}\right] r_1 + \tfrac{2}{3} r_2 \mapsto$$

$$\mapsto \left[\begin{array}{cc|cc} 2 & 0 & \tfrac{4}{3} & \tfrac{2}{3} \\ 0 & \tfrac{3}{2} & \tfrac{1}{2} & 1 \end{array}\right] \begin{matrix} \tfrac{1}{2} r_1 \\ \tfrac{2}{3} r_2 \end{matrix} \mapsto \qquad \left[\begin{array}{cc|cc} 1 & 0 & \tfrac{2}{3} & \tfrac{1}{3} \\ 0 & 1 & \tfrac{1}{3} & \tfrac{2}{3} \end{array}\right] = \left[I \mid A^{-1}\right].$$

Matice A je tedy regulární a platí $A^{-1} = \frac{1}{3}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

Nyní řešení stejného problému pomocí appletu:

1. Do textových polí nejdříve zadáme matici, kterou chceme počítat a poté zmáčkneme tlačítko *Výpis matice*.
2. Dalším krokem je přičtení poloviny prvního řádku k druhému řádku. To provedeme vyplněním hodnoty „$1/2$“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého na index druhého řádku.
3. Dále je přičtení dvou třetin řádku druhého k prvnímu řádku. To provedeme vyplněním hodnoty „$2/3$“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého na index prvního řádku.
4. První řádek vynásobíme hodnotou „$1/2$“ tak, že do příslušného políčka operace *Vynásobení* vyplníme hodnotu „$1/2$“ a poté zvolíme index prvního řádku.
5. Druhý řádek vynásobíme hodnotou „$2/3$“ tak, že do příslušného políčka operace *Vynásobení* vyplníme hodnotu „$2/3$“ a poté zvolíme index druhého řádku.
6. Matice je upravena do tvaru jednotkové matice. Ve výstupní textové oblasti je zobrazena získaná inverzní matice.

2.3.5 Kongruence symetrických matic

Pokud je $A = \lfloor a_{ij} \rfloor$ symetrická matice. Pak existuje regulární matice T a diagonální matice D tak, že

$$TAT^T = D.$$

Applet kongruence symetrických matic

Applet je rozložen do dvou částí. V první části (poté co uživatel zadá příslušnou matici) je cílem upravit matici na diagonální tvar. Ještě předtím než jsou započaty jakékoliv výpočty, algoritmus nejdříve otestuje, zda-li je matice symetrická (tedy je-li transponovaná matice shodná s matici původní). K samotnému výpočtu po splnění symetričnosti je možné využít elementární kongruence, tedy elementárních operací, na které navazuje následně ještě jejich sloupcová varianta (například budeme-li chtít vynásobit pěti druhý řádek, vynásobí se po aplikování této operace i druhý sloupec hodnotou pět). Jakmile se podaří matici v diagonálním tvaru získat (matice D), přecházíme do druhé fáze procesu. Tím je aplikace stejných řádkových operací, jaké byly použity při hledání diagonální matice, na matici jednotkovou (matice T). Po úspěšném ukončení této fáze už je na řadě pouze zkouška, ve které musí platit rovnost levé strany $TAT^T = D$ (kde matice T^T je transponovaná matice T).

Příklad:

Hledáme regulární matici T a diagonální matici D, tak aby platilo $TAT^T = D$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} r_1 + r_2 \mapsto \underset{s_1 + s_2}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}} \mapsto \begin{bmatrix} 2 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{matrix} \\ \\ -\frac{1}{2} r_2 \end{matrix} \mapsto \underset{s_2 - \frac{1}{2} s_1 \; s_3 - \frac{1}{2} s_1}{\begin{bmatrix} 2 & 1 & 1 \\ 0 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix}} \mapsto$$

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} r_3 + r_2 \mapsto \underset{s_2 + s_3}{\begin{bmatrix} 2 & 0 & 0 \\ 0 & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 \end{bmatrix}} \mapsto \begin{bmatrix} 2 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Nyní aplikujeme řádkové operace, které jsme před chvílí využili, na jednotkovou matici T

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} r_1 + r_2 \quad \mapsto \quad \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{matrix} r_2 - \frac{1}{2} r_1 \mapsto \\ r_3 - \frac{1}{2} r_1 \end{matrix} \quad \begin{bmatrix} 1 & 1 & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix} r_3 + r_2 \quad \mapsto$$

$$\mapsto \begin{bmatrix} 1 & 1 & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ -1 & 0 & 1 \end{bmatrix} r_3 + r_2 \quad = T$$

Ověříme, že platí $TAT^T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} = D$

Nyní řešení stejného problému pomocí appletu:

1. Do textových polí nejdříve zadáme matici, kterou chceme počítat a poté zmáčkneme tlačítko *Výpis matice*.
2. Dalším krokem je přičtení druhého řádku k prvnímu řádku. To provedeme vyplněním hodnoty „1“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého na index prvního řádku. Automaticky je provedena i sloupcová varianta této operace.

a. Dalším krokem je přičtení mínus poloviny prvního řádku k druhému řádku. To provedeme vyplněním hodnoty „$-1/2$“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého na index druhého řádku. Automaticky je provedena i sloupcová varianta této operace.

b. Dalším krokem je přičtení mínus poloviny prvního řádku k třetímu řádku. To provedeme vyplněním hodnoty „$-1/2$“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého na index třetího řádku. Automaticky je provedena i sloupcová varianta této operace.

3. Dalším krokem je přičtení druhého řádku k třetímu řádku. To provedeme vyplněním hodnoty „1“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého na index třetího řádku. Automaticky je provedena i sloupcová varianta této operace.
4. V této chvíli jsme dostali diagonální matici D. Je provedeno aplikování řádkových operací využitých před chvílí na matici jednotkovou (matici T). Poslední věcí v tomto problému už je jenom zkouška, kdy se $TAT^T = D$.

2.3.6 LDL^T rozklad

Rozklad $LAL^T = D$ je základem efektivních algoritmů pro řešení soustav s pozitivně definitními maticemi, neboť je ekvivalentní rozkladům

$$A^{-1} = L^T D^{-1} L \text{ nebo } A = \tilde{L} D \tilde{L}^T,\ \tilde{L} = L^{-1}.$$

Můžeme tedy s pomocí tohoto rozkladu po rozložení na jednotlivé trojúhelníkové matice vypočítat řešení soustav lineárních rovnic.

Applet LDL^T rozkladu

Applet je rozložen do dvou částí. V první části (poté co uživatel zadá příslušnou matici) je cílem upravit matici spolu s jednotkovou maticí na horní trojúhelníkovou matici (znamená to, že všechny prvky pod diagonálou matice musí být nulové). Ještě předtím než jsou započaty jakékoliv výpočty však algoritmus nejdříve otestuje, zda-li je matice symetrická (tedy je-li transponovaná matice shodná s matici původní). K samotnému výpočtu po splnění symetričnosti je možné využít elementární (řádkové) operace. Jakmile se podaří horní trojúhelníkovou matici získat, můžeme z ní vyčíst dvě matice - matici diagonální (matice D) a z pravé strany matici L. Po úspěšném ukončení této fáze už je na řadě pouze zkouška, ve které musí platit rovnost levé strany $LAL^T = D$ (kde matice L^T je transponovaná matice L).

Příklad:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Upravujeme společně s jednotkovou maticí na horní trojúhelníkovou matici

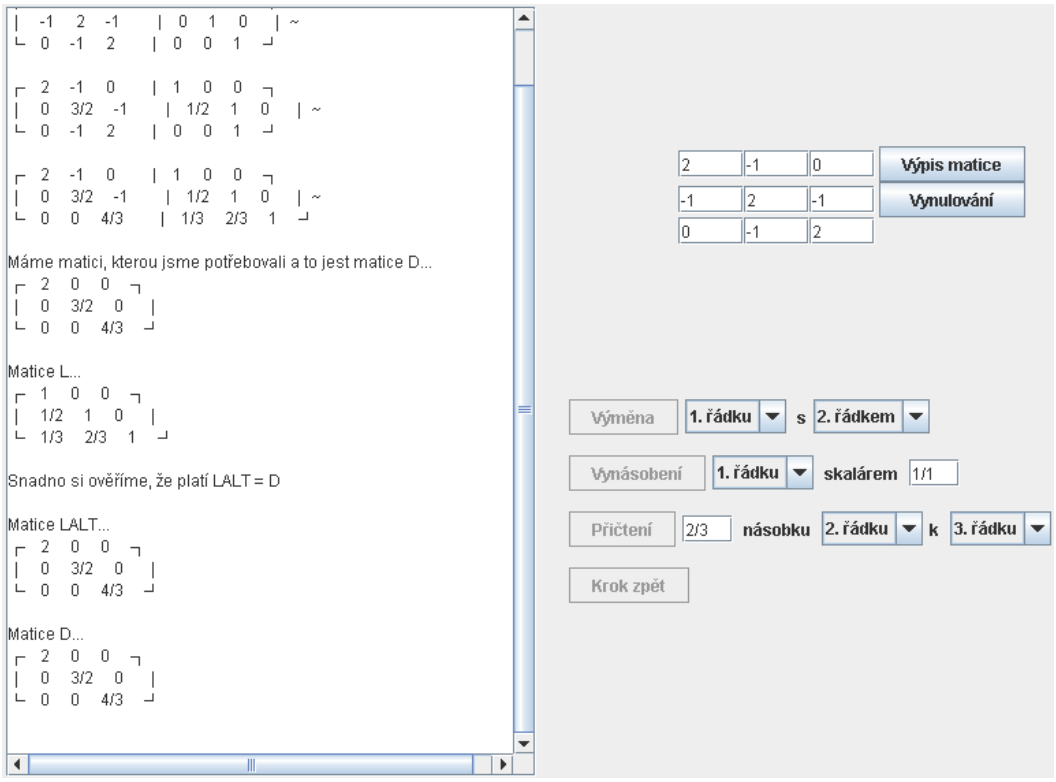
$$\left[A|I\right] = \left[\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 1 & 0 \\ 0 & -1 & 2 & 0 & 0 & 1 \end{array}\right] + \tfrac{1}{2} r_1 \mapsto \left[\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ 0 & \tfrac{3}{2} & -1 & \tfrac{1}{2} & 1 & 0 \\ 0 & -1 & 2 & 0 & 0 & 1 \end{array}\right] + \tfrac{2}{3} r_2 \mapsto$$

$$\mapsto \left[\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ 0 & \tfrac{3}{2} & -1 & \tfrac{1}{2} & 1 & 0 \\ 0 & 0 & \tfrac{4}{3} & \tfrac{1}{3} & \tfrac{2}{3} & 1 \end{array}\right], \quad L = \begin{bmatrix} 1 & 0 & 0 \\ \tfrac{1}{2} & 1 & 0 \\ \tfrac{1}{3} & \tfrac{2}{3} & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & \tfrac{3}{2} & 0 \\ 0 & 0 & \tfrac{4}{3} \end{bmatrix}$$

Platí $$LAL^T = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & \frac{2}{3} & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & 1 & \frac{2}{3} \\ 0 & 0 & 1 \end{bmatrix} = D$$

Nyní řešení stejného problému pomocí appletu:

1. Do textových polí nejdříve zadáme matici, kterou chceme počítat a poté zmáčkneme tlačítko *Výpis matice.*
2. Dalším krokem je přičtení poloviny prvního řádku k druhému řádku. To provedeme vyplněním hodnoty „$1/2$“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index prvního řádku a druhého na index druhého řádku.
3. Dalším krokem je přičtení dvou třetin druhého řádku k třetímu řádku. To provedeme vyplněním hodnoty „$2/3$“ v elementární operaci *Přičtení* a nastavením prvního JComboBoxu na index druhého řádku a druhého na index třetího řádku.
4. V této chvíli jsme dostali diagonální matici D a dolní trojúhelníkovou matici L.
5. Poslední věcí v tomto problému už je jenom zkouška, kdy se $LAL^T = D$.



Závěr příkladu řešeného pomocí appletu LDL^T rozkladu

2.3.7 Determinant matice

Nechť $A = \left[a_{ij} \right]$ je čtvercová matice řádu n s reálnými nebo komplexními prvky. Determinant matice A je číslo, které značíme $\det A$ nebo $\left| A \right|$ a vypočteme jej podle následujících pravidel

(D1) Je-li $n = 1$, pak $\det A = \det \left[a_{11} \right] = a_{11}$.

(D2) Předpokladejme, že $n > 1$ a že umíme určit determinant libovolné čtvercové matice řádu $n - 1$.

Pak $$\det A = a_{11} \left| M_{11}^{A} \right| - a_{12} \left| M_{12}^{A} \right| + a_{13} \left| M_{13}^{A} \right| - \ldots + (-1)^{n+1} a_{1n} \left| M_{1n}^{A} \right| .$$

Elementární řádkové úpravy ovlivňují velmi jednoduše hodnotu determinantu.

1. Přičtení násobku některého řádku k jinému hodnotu determinantu nezmění
2. Vzájemná výměna dvou řádků změní znaménko determinantu
3. Vynásobení řádku skalárem vynásobí tímto skalárem determinant

Elementární řádkové úpravy matice proto můžeme využít k převodu matice na speciální tvar vhodný pro výpočet determinantu. Ve výsledku je determinant trojúhelníkové matice roven součinu prvků hlavní diagonály matice.

Geometrický význam determinantu je například obsah rovnoběžníku (u matic řádu 2), nebo objem rovnoběžnostěnu (u matic řádu 3).

Applet determinantu

Poté co uživatel zadá příslušnou matici, je cílem upravit matici tak, aby byla trojúhelníková (tedy buď dolní troúhelníková matice nebo horní trojúhelníková matice). Jakmile je dosáhnuto trojúhelníkové matice, je spočítán determinant, který ovlivňují i elementární operace využité k výpočtu.

3 Test

3.1 Rozklad problému (MathML)

Námětem na druhou část této bakalářské práce byl test pro uživatele, kteří si chtějí vyzkoušet své znalosti z různých oblastí lineární algebry. Jedním z předpokladů bylo vhodné vytvoření webové stránky s tímto testem. Dalším předpokladem byl náhodný výběr příkladů z datového souboru. A posledním velmi důležitým předpokladem bylo co nejefektivnější zvolení způsobu zobrazování matematických rovnic, vzorců, výrazů apod. na webové stránce.

Nejdříve bych začal od konce a to řešením posledního předpokladu. Navrhovaných možností pro efektivní způsob zobrazování matematických symbolů na webu bylo několik.

Matematické výrazy ve formě obrázků – tedy vkládání matematických výrazů, které se nedají v HTML popsat, do obrázků (nejčastěji formát GIF a PNG) a ty zobrazit jako součást dokumentu. Tento přístup má ale řadu nevýhod:

- při změně velikosti fontu se velikost obrázku nemění
- problém s vhodným řádkováním, odsazování za sebou jdoucích řádků
- kvalita tisku závisí na kvalitě obrázku a neodpovídá kvalitě tisku textu okolo výrazů
- pomalé načítání stránek s velkým počtem obrázků
- v obrázcích není možné vyhledávat
- datová velikost není zrovna ideální
- složitá manipulace, tvorba grafickým editorem
- nemožnost interpretace dat, žádná aplikace z obrázku nevyčte jeho význam
- dynamická tvorba (závislá na datech uživatele) takových vzorců je nemožná - není možné měnit vzorec přímo v kódu

Nejčastěji používané programy v této oblasti jsou například LaTeX2HTML nebo export do HTML v programu Maple.

Například LaTeX2HTML je automatický převodník dokumentu v LaTeXu do formátu HTML. Převodník rozdělí dokument na více částí a automaticky vloží odkazy (linky) mezi tyto části, vytvoří obsah, index, převádí vzorce a vložené obrázky do formátu GIF (PNG) a vkládá je do dokumentu.

Zobrazování matematických výrazů pomocí appletů a pluginů - matematické formule vstupují v TeXu nebo v jiném kódování a jejich zobrazování je zabezpečené pomocí pluginů a appletů. Nevýhodou tohoto přístupu je často závislost na konkrétním prohlížeči či platformě. Vyvstává nutnost instalovat podpůrné programy a často je třeba dlouho čekat na načtení dokumentu.

Například je možno zmínit IBM techexplorer. Plugin pro MSIE a pro Netscape Navigator na nejrozšířenější platformy. Umí přímo interpretovat podmnožinu TeXových (LaTeXových) příkazů a pomocí pluginu zobrazit tyto dokumenty na Webu.

Nebo je možno zmínit WebEQ, který pomocí sady Java-appletů umožňuje do stránek vkládat vzorce zapsané v jednoduché syntaxi, vycházející z typografického systému TeX.

Řešení bez HTML (web jen jako zprostředkovatel) - v tomto případě web slouží jen jako zprostředkovatel odborného dokumentu v některém ze standardních elektronických formátů. Nejčastěji jsou používané formáty PS, DVI a PDF. Výhodou je možnost kontrolovat konečný vzhled dokumentu, což při použití jazyka HTML nemusí být vždy tak jednoduché a možné. Nevýhodou je velikost přenášených souborů, nutnost instalace prohlížeče pro každý z formátů a v některých případech i problémy s nekompatibilitou formátů (jako nejčastěji používaný se v současnosti jeví formát PDF - prohlížeč Acrobat Reader je volně přístupný pro všechny platformy a možnosti formátu PDF a jeho vytváření jsou velmi rozsáhlé).

Ani jedna z těchto možností není dostatečně efektivní a proto byl zvolen k zobrazování matematických vzroců na webu jazyk MathML.

3.1.1 Popis MathML

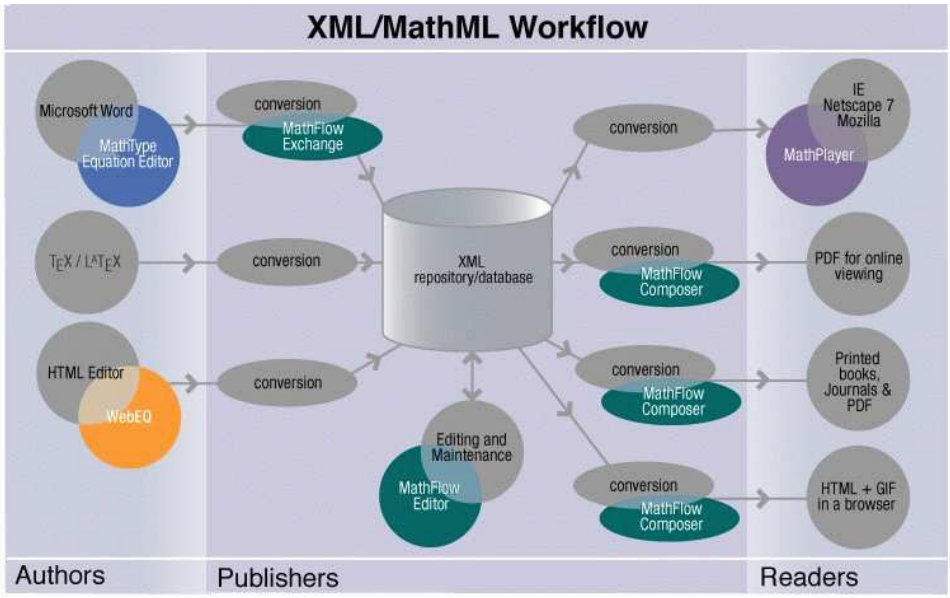
Cílem MathML bylo a je poskytnout platformu, umožňující kódovat matematické výrazy pomocí XML za účelem jejich prezentace v elektronické (primárně na WWW) i tištěné podobě a také výměny matematických informací mezi různými aplikacemi či systémy (nástroje pro tvorbu dokumentů, aplikace pro matematické výpočty a výuku matematiky, systémy správy obsahu aj.).

MathML je vytvořeno tak, aby splňovalo několik základních požadavků na možnost prezentace matematiky na Web. Prvním požadavkem je možnost vkládání MathML do HTML takovým způsobem, aby bylo přístupné budoucím prohlížečům, vyhledávačům a většině aplikací, které nyní manipulují s HTML. Druhým požadavkem je zpřístupnění MathML v dnešních prohlížečích takovým způsobem, aby informace byla nějakým způsobem pro čtenáře rozluštitelná. Posledním z hlavních požadavků na MathML je možnost konverze existujících dokumentů s matematikou v jiném značkovacím jazyce do MathML.

V současnosti si klade jazyk MathML tyto cíle:

- kódování matematických dokumentů pro výuku a vědeckou komunikaci na všech úrovních;
- kódování matematických vzorců i zachycení matematické sémantiky;
- ulehčení konverze z a do jiných matematických formátů na úrovni prezentační a obsahové. Mezi požadované výstupy patří grafický výstup, zvukový výstup, vstup pro počítačové systémy pro algebru, ostatní jazyky pro formátováni matematiky, jako je TeX, zobrazení čistého textu a tištěný výstup včetně Braillova písma;
- podporu efektivního vyhledávání v delších výrazech;
- podporu pro rozšiřitelnost;
- dobrou čitelnost pro uživatele a jednoduchost strojového generování a zpracování.

K dispozici jsou dvě sady tagů, jedna pro prezentaci matematického obsahu (prezentační elementy - popisují matematickou formu zápisu (strukturu) matematického výrazu a určují, jak bude kódovaný výraz v prohlížeči zobrazen), druhá pak pro vyjádření jeho významu (sémantické elementy - přímo matematické objekty, na rozdíl od jejich prezentace). Popis sémantiky zde přidává oproti dřívějším jazykům pro zápis matematických formulí (TeX apod.) nové možnosti zpracování matematického obsahu - příkladem zde může být výměna informací s různými matematickými aplikacemi, indexování a prohlížení obsahu na WWW (sémantický Web) či zpřístupnění obsahu matematických výrazů vizuálně postiženým osobám.



Příklad schématu, naznačujícího způsob nasazení MathML v rámci řešení MathFlow

3.1.2 Podpora MathML v internetových prohlížečích

Podpora MathML v prohlížečích je nyní už poměrně dostatečná. Velmi dobře je na tom Amaya (testovací prohlížeč W3C), která umožňuje zobrazení i editaci vzorce. V dnešní době přímo podporují formát MathML současné verze Mozilly a její klony. MSIE nepodporuje MathML, je však možno podporu zařídit přidáním externího MathPlayer pluginu. Opera (od verze 9.5) podporuje MathML pro CSS, ale to je neschopné vhodně zobrazovat pozici diakritických znamének (před verzí 9.5 byl požadovan JavaScript, který emuloval MathML podporu).

- Windows:
  - MSIE 5.0 s nainstalovaným TechExplorer pluginem
  - MSIE 5.5 s nainstalovaným MathPlayer nebo TechExplorer pluginem
  - MSIE 6.0+ volitelně s nainstalovaným MathPlayer nebo TechExplorer pluginem
  - Netscape 6.1 s nainstalovaným TechExplorer pluginem
  - Netscape 7.0+
  - Amaya, všechny verze (pouze prezentační MathML)
  - Mozilla 0.9.9+
- Macintosh:
  - MSIE 5.0+ s nainstalovaným TechExplorer pluginem
  - Mozilla 0.9.9+
- Linux/Unix:
  - Netscape 6.1 s nainstalovaným TechExplorer pluginem
  - Netscape 7.0+
  - Mozilla 0.9.9+
  - Amaya, všechny verze (pouze prezentační MathML)

Jelikož však v programu jsou načítány matematické vzorce z XML dokumentu do prostředí .NET a pravá podpora jazyka MathML v jednotlivých prohlížečích je založena na skutečném XHTML nebo XML, nastává jeden problém. Pokud pošleme stránku jako obyčejné HTML, tedy *text/html*, parser to nezpracuje jako X(HT)ML, ale prostě jako HTML a tam pochopitelně jmenné prostory nepozná a bude je ignorovat, respektive vykreslí obsah jednotlivých elementů nezávisle na tom, co v tom jmenném prostoru znamenají. Dalším problémem je, že prohlížeč může sice znát XHTML, ale pokud nezná MathML, vykreslí prostě obsah elementů za sebou, bez dalšího fyzického formátování (jako by byly například ve spanu). Pokud prohlížeč nezná požadovaný jazyk, neexistuje spolehlivá možnost, jak mu říct, aby nic nevykreslil, případně aby vykreslil alternativní obsah jako v případě obrázku.

Pro oba hlavní internetové prohlížeče, pro které je program optimalizován (tedy pro MSIE a Mozilla Firefox) je nutné doplnit kód, který vykresluje stránku, o pár nových řádků.

- Doplnění definice dokumentu o elementy, které využívá jazyk MathML

  ```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
        "xhtml- math11-f.dtd">
  ```

- Doplnění jmenného prostoru ukazujícího na XHTML o jazyk MathML

  ```
  <html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:m="http://www.w3.org/1998/Math/MathML">
  ```

- Po nainstalování pluginu MathPlayer je nutné vložit objekt rozeznávající již nainstalovaný plugin MathPlayer:

```
<object id="MathPlayer"
        classid="clsid:32F66A20-7614-11D4-BD11-00104BD3F987"></object>
<?import namespace="m" implementation="#MathPlayer"?>
```

- K rozeznání, že jde o matematické výrazy jazyka MathML a že je nutno je správně graficky zobrazovat, je do hlavičky stránky přidán volně šířitelný skript (Příloha 7.3).

- Do těla zdrojové kódu je jako poslední věc nutno přidat odstartování skriptu, který zajišťuje v případě výskytu MathML kódu jeho zpracování

```
<body onload="convert()">
```

3.1.3 Nástroje

MathML je už podporováno ve většině velkých matematických systémů. V systému Mathematica je podpora zajištěna od verze 4.0. Tato podpora znamená možnost importu i exportu. Rovněž Maple přistoupil k podpoře MathML už ve verzi 7.

MathML je také podporován hlavními kancelářskými produkty jako například OpenOffice, KOffice a Microsoft Office 2007. Microsoft Word 2007 umí přes schránku obousměrně pracovat i s formátem MathML, takže můžete vzorec vytvořit i pomocí jiného nástroje, který umí s MathML pracovat a pak ho do Microsoft Word 2007 vložit. W3C prohlížeč/editor Amaya může také být použit jako wysiwyg MathML editor. Dále stojí za zmínku ještě například méně populární editor MathML Equation Editor.

Pro tvorbu matematických vzorců jsem však vyzkoušel dva známé a kvalitní editory, které dokáží vyprodukovat zdrojový kód v jazyce MathML – MathType 6 a WebEQ 3.7.3.

3.1.3.1 WebEQ

Na výběr je zde ze tří možností jaký bude formát používaný v matematických výrazech. WebEQ je připraveno pro tři hlavní způsoby zobrazení výrazů: obrázky, MathML a HTML applety.

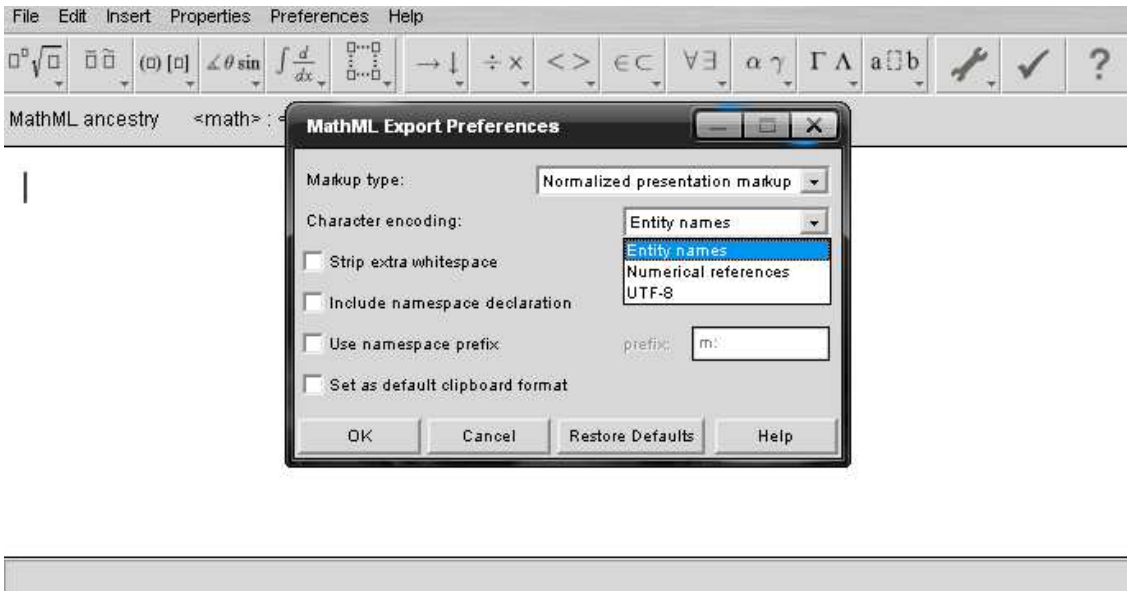
- Obrázky – tam, kde je pohromadě hodně výrazů, nebo stránky kde interaktivita je zbytečná jsou nejlepší volbou obrázky nebo MathML. Obrázek je tedy nejuniverzálnějším řešením pokud potřebujeme aby se výrazy zobrazovaly správně na všech platformách bez potřeby aktuálních prohlížečů a nainstalované podpory pro MathML. Obrázky se jednoduše tvoří pomocí WebEQ nástrojů. Volíme mezi PNG nebo JPG formátem a takto uložené obrázky potom vkládáme v HTML editoru pomocí tagu <*img*>. Výhody a nevýhody tohoto formátu jako interpreta matematických výrazů už bylo popsáno výše.

- MathML - obsahuje více informací o výrazu, jeho struktuře a významu než ostatní formáty, proto je nejlépe použít MathML všude, kde je to možné. Hlavní limitou je podpora MathML v prohlížečích. MathML přislibuje nejrychlejší a nejkvalitnější zobrazení a tisk. Obsahuje mnoho informací o výrazu a tak jsou výrazy použitelné i v dalších souvislostech.

- Viewer Control aplety - pokud je interaktivita naším hlavním cílem, jsou WebEQ Viewer Control applety nejlepší volbou. Applety jsou zvláště multiplatformním prostředím. Nabízejí spolupráci s JavaScript API pro pokročilé programování prezentací. Nevýhodou jsou velké nároky na výpočetní výkon počítače a proto je doporučováno mít pohromadě pouze několik appletů. Jak WebEQ Editor tak Publisher nabízejí jednoduché generování HTML kódu apletů pro snadné vložení do kódu naší prezentace. Applety nejlépe fungují v MSIE verze 5.0 a Netscape 4.0 a vyšších.

Tvoření jednotlivých matematických výrazů ve WebEQ si můžeme představit podobně jako při práci s tvořením vzorců v Microsoft Office. Jediný rozdíl je v tom, že máme speciální nastavení pro výstup z tohoto programu, které nabízí i několik možností právě pro jazyk MathML (čímž si můžeme usnadnit spoustu věcí). Je to například použití jakéhokoli namespace prefixu, kódování znaků, o jaké elementy v rámci MathML jde (jestli prezentační nebo sémantické) a podobně.

Výsledný zdrojový kód dostaneme v menu klasickým uložením do souboru MathML. Na zdrojový kód se můžeme podívat otevřením takto uloženého souboru v textovém editoru (např. poznámkový blok).



Menu pro export do jazyka MathML v programu WebEQ 3.7.3

3.1.3.2 MathType

MathType 6 umožňuje vytvořit snad jakýkoli vzorec a ten následně uložit ve formátu Encapsulated PostScript, GIF a Windows Metafile. Dále je možný překlad do TeXu a MathML (verze 1.0 i 2.0). Tento překlad probíhá velice jednoduše - kopírováním. Vzorec v MathType kopírujeme a následně vkládáme do HTML editoru (už jako MathML nebo jako TeX). Tato možnost se nastavuje v nabídce *Preferences* | *Translators*.

Nešikovné je, že MathType nepodporuje opětovné vložení kódu MathML a jeho přeložení do původního vzorce (to samé platí o TeXu). Nedostatky postihly také pole formátování. MathType umožňuje barvit a formátovat (dle významu funkce, proměnná a podobně), styl však není možno exportovat do MathML. To plyne nejspíš z používání obecného formátu XML, ale stejně by mohla existovat možnost exportu stylů do dokumentů HTML v podobě CSS. Je třeba dodat, že MathType slouží pouze k tvorbě a úpravě vzorců, ne k vyhodnocování.

Po nainstalování MathType 6 dojde k automatickému propojení se sadou Microsoft Office, kde se zobrazí nový panel nástrojů určený pro tvorbu vzorců.

S touto pomocí je možné exportovat dokument do XHTML + MathML nebo do HTML + GIF. V nabídce *Vložit* | *Objekt* se objeví nová položka s názvem MathType 6. Sice zde stále zůstává Equation Editor 3.0, avšak po jeho spuštění dojde k otevření MathType 6.

Více o nastavení MathType 6 pro vytvoření kódu v jazyce MathML a vytváření příkladů – viz Příloha 7.2.

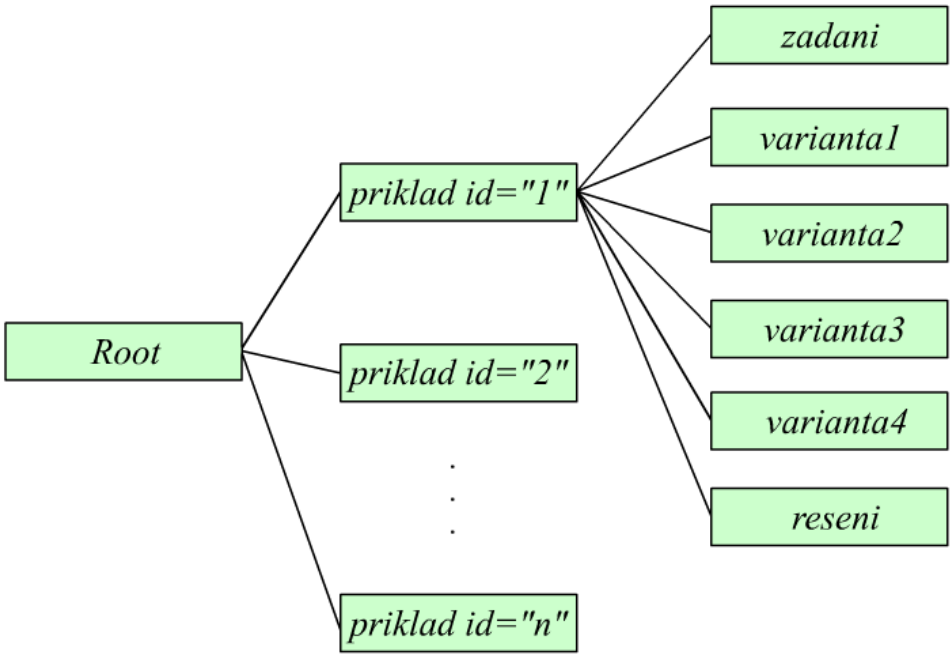
3.2 Rozklad problému (XML)

Další důležitou věcí, která byla zapotřebí řešit bylo vhodné zvolení uložiště zdrojových kódu v jazyce MathML, ze kterého by se následně dala tato data jednoduše vybírat a předávat internetovému prohlížeči. Jednou ze dvou možností, která mě v tomto problému napadla je využití databáze. Z důvodu, že je ale jazyk MathML tak úzce svázan se značkovacím jazykem XML a sám z něho vychází a je na jeho principech založen, rozhodl jsem se ukládat data ve formě MathML právě do XML dokumentů. S XML dokumenty je jednoduchá práce a při navrhnutí vhodné struktury dokumentu je zachována i naprostá přehlednost. Více o XML, navrhnuté struktuře, implementování vytvořeného matematického výrazu v MathType 6 do XML dokumentu – viz Příloha 7.2.

3.3 Rozklad problému (Prostředí pro vývoj webové aplikace)

Tímto se dostáváme k poslednímu předpokladu pro úspěšné vytvoření požadovaného testu. Bylo potřeba spojit XML dokumenty s příklady obsahujícími MathML jazyk pro zobrazování matematických vzorců s nějakým prostředím, které bude umožňovat programovatelný přístup k těmto datům, které bude schopno vytvořit webovou stránku v požadovaných formátech a s požadovaným grafickým výstupem, a která bude pro uživatele snadno ovladatelná. Je více jak zřetelné, že klasické využití pouze jazyka HTML je nereálné. Určitě by byla dobrá možnost komunikace jazyka HTML a PHP ve spolupráci s nějakou databází. Jak už ale bylo naznačeno dříve, z hlediska vývinu nových technologií a starnutí předešlých, z hlediska výhod prostředí .NET a jeho velké spolupráci s XML dokumenty a z hlediska velkým doporučení ze stran zkušenějších odborníků v této oblasti, bylo zvoleno prostředí .NET. Pro vytvoření testu bylo využito MS Visual Studia 2008 a jeho profesionální edice, která je pro studenty VŠB skrze volně dostupnou licenci volně ke stažení a tím skvěle dostupná. Samotná práce (vkládání komponent či práce s XML dokumentem) v tomto prostředí probíhala v programovacím jazyce C#, který je velice podobný jazyku Java (tato podobnost a dřívější zkušenosti s jazykem Java byli hlavním důvodem právě pro toto zvolení před například jazykem Visual Basic).

.NET framework nám nabízí několik nástrojů, které s XML umí pracovat. Každý z nich se hodí v jiné situaci. Já využil třídu XmlDocument, která používá přístup DOM. Tedy na počátku je otevřen XML dokument. Zavoláním metody Load je XML soubor jednoduše načten do paměti. Náš objekt, který je instancí třídy XmlDocument, si XML soubor přečte a vytvoří jakýsi virtuální strom elementů, který si můžete představit v našem případě takto:



Prochází se všechny objekty XmlNode (každý XmlNode je vlastně jeden obdélníček na obrázku), které vyhovují dotazu */Root/priklad*. Tento dotaz vybere všechny elementy *priklad*, které najde na cestě *Root***,** tedy uvnitř kořenového elementu *Root***.**

V případě všeobecného testu na toto načtení navazuje výběr náhodného příkladu z tohoto stromu (určitý element *priklad*) a ten je dále poslán ke zpracování. Tento náhodný výběr je založen na prohledání posledního příkladu v daném stromu a zjištění jeho *id*. Poté dojde k zavolání funkce, která náhodně vybere číslo menší nebo rovno číslu zjištěného identifikačního čísla a k následnému výběru. V případě testu, který je zaměřen na jednotlivé kapitoly je opět tento dokument načten, ale poté nedochází k žádnému výběru, nýbrž je dále ke zpracování poslán celý strom daného XML dokumentu.

Zobrazení probíhá tak, že k pěti předem nadefinovaným RadioButtonům je vypsán pomocí metody pro práci s XML dokumentem obsah (ať už náhodně zvoleného nebo vybraného příkladu), který je mezi elementy *zadani* a elementy *varianta1 – varianta4*. V případě všeobecného testu je zobrazeno pět příkladů z různých oblastí lineární algebry (matice a maticové operace, výpočet inverzní matice, LU rozklad, kvadratické formy a determinanty). Pokud se jedná o test zaměřený na určitou kapitolu lineární algebry, pak jsou zobrazeny všechny příklady uložené v příslušném dokumentu.

Po zobrazení testu, vyplněním testu uživatelem a stisknutí tlačítka pro vyhodnocení testu se dostane test do další fáze, ve které se test přepne na další stránku. Jelikož si algoritmus stále pamatuje příklady, které zobrazoval v rámci vyplňování testu na první stránce, vypíše je znovu. Navíc dochází k tomu, že je srovnáván u každého příkladu index zvoleného RadioButtonu s hodnotou uloženou v obsahu elementu *reseni* (hodnota řešení je ve formě datového typu integer a nabývá hodnot 1-4 podle toho, která z variant 1-4 je správná). Na to navazuje vybarvení daných RadioButtonů podle toho jaká je odpověď.

- Pokud je odpověď správná je vybarvení provedeno zelenou barvou.
- Pokud je odpověď nesprávná je vybarvení provedeno červenou barvou a správná odpověď je provedena šedou barvou.
- Pokud nebyla zadána žádná odpověď je zobrazena pouze ta odpověď, která je správná a to šedou barvou.

Na závěr je provedeno vyhodnocení testu (tedy kolik z celkového počtu příkladů bylo správně zodpovězeno a kolik jich bylo zodpovězno nesprávně (pokud uživatel na nějakou odpověď neodpověděl, je tato odpověď brána jako nesprávná)).

Kontrolní test je dále rozšířitelný v jakýchkoliv směrech. Je možno rozšířit kterýkoliv z testů do rozměrů desítek až stovek příkladů. Podobně je na tom i množství příkladů přímo v jednom testu, kde je možné počet příkladů zvýšit na mnohem větší počet. Úpravy se v prvním případě týkají pouze přídávání dalších příkladů do XML dokumentu (webová aplikace je naprogramována na obecný počet příkladů uvnitř jednoho dokumentu – viz Příloha 7.2), v druhém případě jde pouze o změnu několika řádků zdrojového kódu a vytvoření nové verze webové aplikace (nový build).

4 Webová stránka pro uživatele

4.1 Applet na WWW stránce

Před spuštěním appletu je třeba na něj umístit odkaz do HTML dokumentu tvořícího WWW stránku. To se děje pomocí speciální formátovací značky <APPLET>, která má obecně následující strukturu:

```
<APPLET
   CODE = " třída.class "
   WIDTH = " šířka "
   HEIGHT = " výška "

   [ NAME = " jménoInstanceAppletu "]
   [ CODEBASE = " url "]
   [ ARCHIVE = " JARsoubor1,  JARsoubor2, ...,  JARsouborN "]
   [ ALT = " alternativníText "]
   [ ALIGN = " zarovnávání "]
   [ HSPACE = " vodorovnéOdsazení "]
   [ VSPACE = " svisléOdsazení "] >

[ <PARAM NAME = " jméno1 " VALUE = " hodnota1 "> ]
[ <PARAM NAME = " jméno2 " VALUE = " hodnota2 "> ]
  ...
[ <PARAM NAME = " jménoN " VALUE = " hodnotaN "> ]
[ AlternativníHTML ]
</APPLET>
```

Značka <APPLET> je rozdělena na tyto základní části:

Atributy appletu - jsou uzavřeny v prvních ostrých závorkách.

Parametry appletu - každý parametr PARAM má jméno (NAME) a hodnotu (VALUE). Ke čtení parametrů slouží metody *getParameter()* a *getParameterInfo()*.

AlternativníHTML - je HTML text zobrazený prohlížečem, který nespouští applety.

Jednotlivé atributy značky <APPLET> mají tento význam:

- CODE = třída.class - udává jméno hlavní třídy appletu.

- WIDTH = šířka - udává šířku grafického výřezu (v pixelech) na WWW stránce, kam může applet zobrazovat.

- HEIGHT = výška - udává výšku grafického výřezu (v pixelech) na WWW stránce, kam může applet zobrazovat.

- NAME = jménoInstanceAppletu - udává jméno instance appletu, na který je pak možné odkazovat metodou *getApplet()* (slouží pro komunikaci appletů).

- CODEBASE = url - udává absolutní nebo relativní URL, kde se budou hledat třídy appletu a JAR soubory (není-li použito, je hodnotou CODEBASE URL adresáře, odkud byl načten HTML soubor odkazující na applet).

- ARCHIVE = JARsoubor1, JARsoubor2, ..., JARsouborN - specifikuje jeden nebo více komprimovaných souborů, které se budou nahrávat při načtení HTML dokumentu. Tyto archívy mohou obsahovat .class soubory a další data appletu. Jejich použitím se výrazně zkrátí celková doba stahování ze sítě.

- K souborům z JAR archívů se z programu přistupuje jako by se nalézaly v adresáři relativně od CODEBASE.

- ALT = alternativníText - obsahuje text, který prohlížeč zobrazí v případě, že rozeznává značku <APPLET>, ale nepodporuje grafický režim.

- ALIGN = zarovnávání - určuje horizontální zarovnávání, může nabývat některé z hodnot: left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom.

- HSPACE = vodorovnéOdsazení - udává vodorovné odsazení grafického výřezu appletu na WWW stránce (v pixelech).

- VSPACE = svisléOdsazení - udává svislé odsazení grafického výřezu appletu na WWW stránce (v pixelech).

Příklad jednoduchého odkazu jednoho z appletů umístěného na webové stránce:

```
<applet code="Gauss3.class" codebase="gauss33/bin/" width=800 height=600>
</applet>
```

4.2 Umístění webové aplikace s testem na server

MS Visual Studio 2008 velice usnadňuje získání výsledné webové aplikace. Tvoření kódu je možno zakončit vypracováním (buildem) webové stránky a tím dostat hotovou webovou aplikaci v místě uložení projektu. Tato složka, ve které se nalézají veškeré soubory potřebné ke správnému chodu aplikace, je poté zkopírovatelná na jakýkoliv server, který podporuje technologie ASP.NET (v našem případě jeho verzi 3.5). Problém může nastat ve chvíli, kdy například k naší webové aplikaci využíváme přístup k datům databáze (pak je nutné přenastavit soubor *web.config* – to se však tohoto projektu netýká).

5 Závěr

Bakalářská práce si kladla za cíl vytvořit funkční, uživatelsky jednoduchou a přehlednou webovou stránku, která bude schopna ostrého provozu a bude splňovat všechny funkce snadnější výuky, které byly zadány.

Webová stránka musela projít celým cyklem vývoje. Musely být nejprve specifikovány požadavky důležité pro pozdější tvorbu appletů a pro jejich správnou interpretaci. V návrhu implementace byly navrhnuty postupy, které vedly ke splnění všech funkcionalit. Dále musel být vybrán nejvhodnější postup pro zobrazování matematiky na webu ke graficky správnému zobrazování kontrolních testů a vhodná spolupráce s technologií pro vývoj webových aplikací.

Vznikl tak celek, který splnil všechny požadavky. Obsáhuje možnosti pro snadnější praktickou výuku uživatelů v různých oblastech lineární algebry, včetně možnosti získat teoretické základy.

6 Literatura

[1] *Prof. RNDr. Zdeněk Dostál, DSc.*
LINEÁRNÍ ALGEBRA, skripta
Dotisk 1. vydání, VŠB – TU Ostrava, Ostrava, 2004

[2] **Výuka lineární algebry**
http://vondrak.am.vsb.cz/la-it/, duben 2009

[3] **Internetové technologie**
http://www.interval.cz, duben 2009

7 Příloha

7.1 Příklad zaokrouhlování skrze datový typ „double“ a třídu „Zlomek“

```
/* do proměnné vysledek ukládáme výsledek dělení */
double vysledek = 1.0/7.0;

/* při každé iteraci cyklu se do této proměnné přičte výsledek dělení */
double soucet = 0.0;

/* do proměnné soucet ukladáme sedmkrát výsledek předchozího dělení */
for (int i = 0; i < 7; i++)
{
  soucet += vysledek;
}
```

Pokud v této chvíli vypíšeme výsledek proměnné *soucet*, zobrazí se nám tato hodnota

0.9999999999999998

Kdybychom chtěli získat ze stejného algoritmu opět hodnotu, ale za využití objektu *Zlomek*, mohlo by to vypadat takto:

```
/* vytvoření nových objektů třídy Zlomek */
Zlomek vysledek = new Zlomek (1, 7);
Zlomek soucet = new Zlomek (0, 1);

for (int i = 0; i < 7; i++)
{
  soucet = soucet.plus (vysledek);
}
```

Za předpokladu, že máme naprogramovánu třídu pro tvorbu zlomků a metodu pro sčítání zlomků, můžeme poté očekávat takovýto výstup

1

7.2 Přidávání příkladů do XML dokumentů

7.2.1 Struktura XML dokumentu

První věcí, na kterou je potřeba se podívat, je navrhnutá struktura XML dokumentu a pár základních pravidel při práci s těmito dokumenty, které je nutno dodržovat pro správnou a bezchybnou funkčnost při následném průchodu zdrojovým kódem webové stránky a zobrazení v samotném prohlížeči. Každý XML dokument musí mít právě jeden kořenový element, kterým dokument začíná a na závěr i ukončuje (v našem případě se jedná o element „*<Root>*“ a „*</Root>*“). Další případ, který může způsobit chybu a nesprávné budoucí zobrazení je případ, kdy neprázdné elementy nebudou ohraničeny startovací a ukončovací značkou (vždy musí být tag, který je jednou odstartován a následně v příslušné části také ukončen). Z dalších pravidel je pro nás ještě důležité si dávat pozor na velká a malá písmena, totiž jména elementů rozlišují malá a velká písmena (např. „*<Příklad>*“ a „*</Příklad>*“ je pár, který vyhovuje správně strukturovanému dokumentu, pár „*<Příklad>*“ a „*</příklad>*“ je chybný). Poslední věc ze syntaxe XML, která může nastat - elementy mohou být vnořeny, ale nemohou se překrývat (to znamená, že každý (ne kořenový) element musí být celý obsažen v jiném elementu).

Nyní k mnou navrhnuté struktůře XML dokumentu. Každý příklad se skládá ze čtyř částí:

1. celý příklad je vložen mezi element ‘priklad’, který ve svém startovacím tagu navíc ještě obsahuje atribut ‘id’, jehož hodnota musí být uzavřena v uvozovkách. Tento atribut udává pořádí příkladu v rámci celého XML dokumentu (tento atribut je v dokumentu obsažen pouze pro informativní účel, zdrojový kód se na tento atribut žádným způsobem neodkazuje a ani žádným způsobem s ním nekomunikuje). Tento tag neobsahuje žádný čistý text ani nevykonává žadné operace, pouze označuje na začátek příkladu, kde má započít výběr dat. Všechny další části jsou vnořené elementy pod tento nadelement a celý příklad je uzavřen ukončovacím tagem.

   ```
   <priklad id="1">
   …
   </priklad>
   ```

2. každý příklad začíná zadáním. Do zádání je vkládán čistý text popisující co přesně má uživatel v daném příkladu řešit. Je samozřejmě možné kombinovat text s vkládáním matematických výrazů, rovnic a textů (popis jak tohoto dosáhnout je uveden níže).

   ```
   <zadani>
   …
   </zadani>
   ```

3. následuje první až čtvrtá varianta správného řešení. Pro všechny možnosti řešení lze použít opět jak čistý text, který má uživateli co nejvíce přiblížit tuto možnost řešení, nebo matematické vzorce.

```
<varianta1>
…
</varianta1>
<varianta2>
…
</varianta2>

<varianta3>
…
</varianta3>

<varianta4>
…
</varianta4>
```

4. poslední částí je řešení. Do elementu řešení se vkládá pouze celé číslo (tedy datového typu integer), které je rovno nebo větší jedné a zároveň rovno nebo menší čtyřem ($1 \leq x \leq 4$). Toto číslo od jedné do čtyř udává, která ze čtyř variant možných řešení je ta správná (tedy je-li správné řešení uvedeno ve variantě číslo jedna, bude v řešení uvedeno číslo jedna; je správné řešení varianta tři, bude v řešení číslo tři, atd.).

```
<reseni>
…
</reseni>
```

Vzorově tedy vypadá každý příklad v navrhnuté struktuře XML dokumentu takto:

```
<priklad id="">
  <zadani>
  …
  </zadani>
  <varianta1>
  …
  </varianta1>
  <varianta2>
  …
  </varianta2>
  <varianta3>
  …
  </varianta3>
```

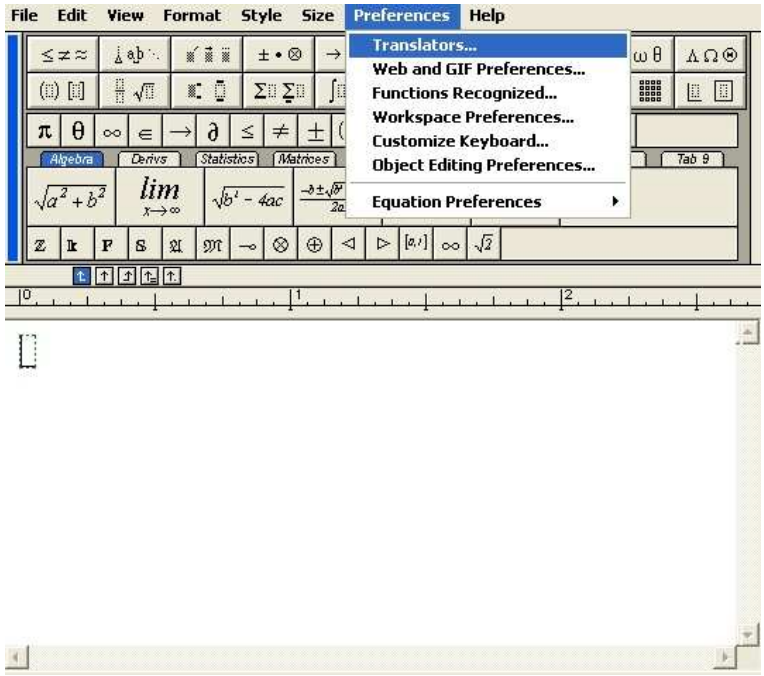
```
    <varianta4>
    …
    </varianta4>
    <reseni>
    …
    </reseni>
</priklad>
```

7.2.2 Tvorba vzorců a jejich vkládání do XML dokumentu

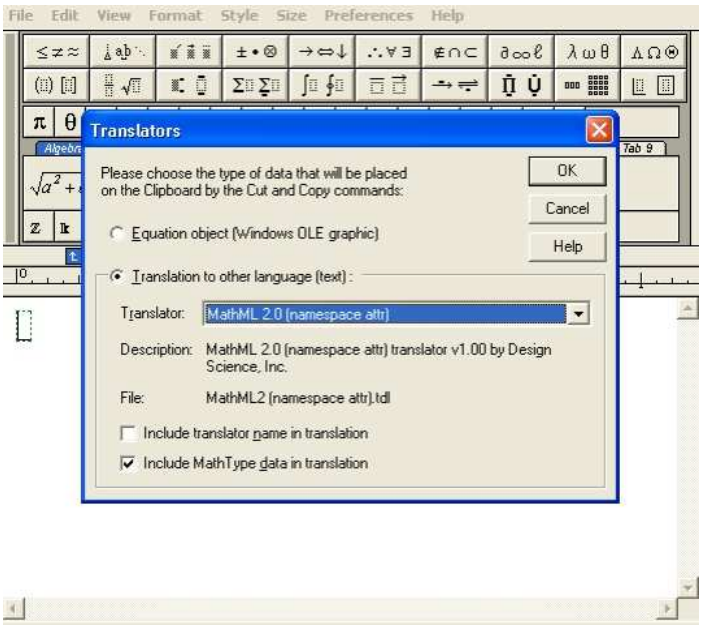
Výše byla popsána struktura XML dokumentu, do kterého je potřeba vhodným způsobem vkládat a doplňovat data. Velmi důležitou věcí je v tomto případě tvorba matematických vzorců a vkládání těchto vzorců na příslušná místa. Pro správné a graficky pěkné zobrazení webové stránky, která obsahuje matematicke vzorce, texty a výrazy je vhodné si zvolit správnou technologii. K našemu cíli nám nejvíce poslouží jazyk MathML, který se připojuje k jazyku HTML a interpretuje matematické vzorce v prohlížeči. V následujícím návodu jak tvořit matematické vzorce a jak je vkládat a upravovat po vložení do XML dokumentu je použit interaktivní nástroj MathType. Tento nástroj dává za možnost jednoho ze svých výstupních formatů právě data v jazyce MathML.

Nyní k návodu:

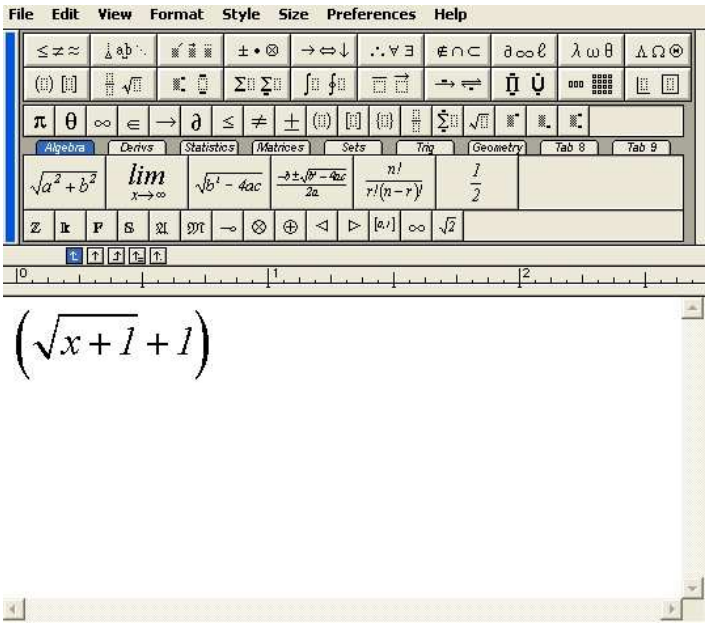
Po otevření aplikace MathType je potřeba udělat menší změnu v nastavení. V hlavním menu v záložce *Preferences* je odkaz na *Translators*.



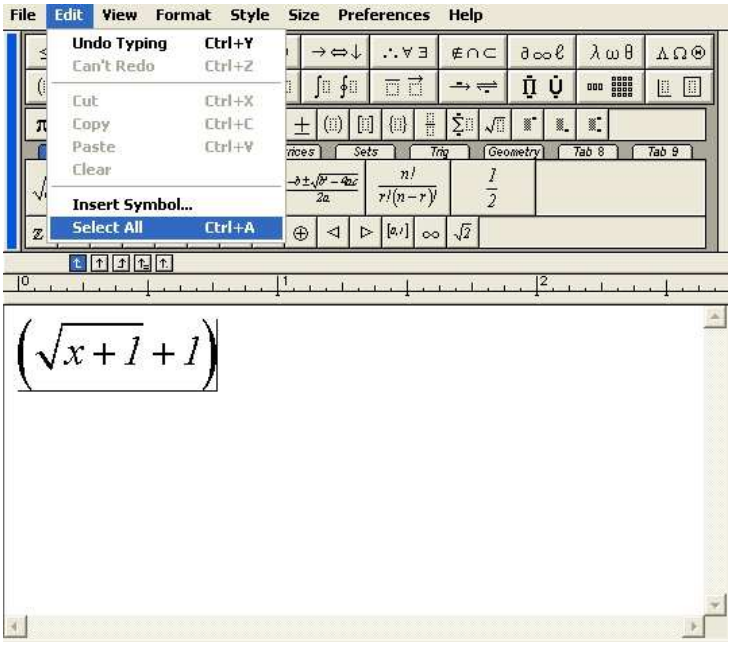
Klikneme na tento odkaz a vyskočí okno, které je potřeba přenastavit podle následujícího obrázku. Jde o to jaký formát dat se uloží do schránky při provedení operace *„Kopírovat"*. Protože chceme, aby náš vzorec byl v jazyce MathML, zvolíme tuto možnost z nabídky.

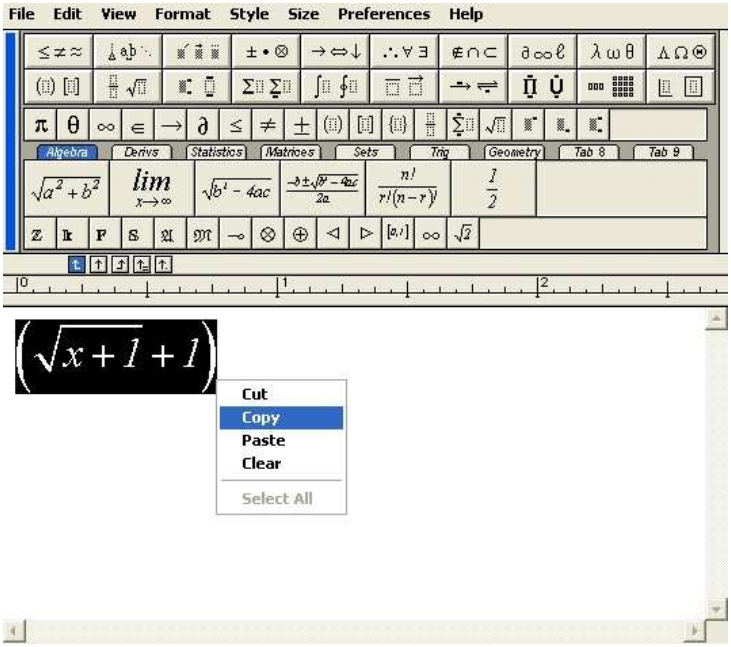


Vše je připraveno, je možné si vytvořit co potřebujeme, např. výraz $\left(\sqrt{x+1}+1\right)$.



Převod do jazyka MahML z aplikace MathType probíhá (jak už bylo zmíněno) přes schránku a to operacemi *„Kopírovat”* a *„Vložit”*. Je tedy potřeba v MathTypu z hlavního menu ze záložky *„Edit”* provést příkaz *„Select Al”* (nebo klávesovou zkratkou CTRL+A) a následně z rozbalovací nabídky námi vybraného vzorce (otevření pravým tlačítkem myši) provést operaci *„Copy”* (nebo klávesovou zkratkou CTRL+C).





Následuje fáze vložení dat do XML dokumentu. Dejme tomu, že výraz, který jsme vytvořili je první varianta.

```
<priklad id="1">
  <zadani></zadani>
  <varianta1>
    <math display='block' xmlns='http://www.w3.org/1998/Math/MathML'>
      <semantics>
        <mrow>
          <mrow>
            <mo>(</mo>
            <mrow>
              <msqrt>
                <mrow>
                  <mi>x</mi>
                  <mo>+</mo>
                  <mn>1</mn>
                </mrow>
              </msqrt>
              <mo>+</mo>
              <mn>1</mn>
            </mrow>
            <mo>)</mo>
          </mrow>
        </mrow>
        <annotation encoding='MathType-MTEF'>MathType@MTEF@5@5@+=feaag
      </semantics>
    </math>
  </varianta1>
  <varianta2></varianta2>
  <varianta3></varianta3>
  <varianta4></varianta4>
  <reseni></reseni>
</priklad>
```

Pro editaci XML dokumentu je použit textový editor PSPad, který usnadňuje velmi kvalitně práci a manipulaci s tímto dokumentem. S vložením obsahu schránky do dokumentu se s potřebnými daty vložilo i pár nepotřebných řádků, které můžeme odstranit a ponechat tak čistý kód jazyka MathML. Zbytečný je párový tag „*semantics*" (potřeba osdstranit jak začínající tak ukončující tag) a dále párový tag „*annotation*". Stejně tak atributy u startovacího elementu „*math*" je pro správné zobrazení potřeba smazat. Po úpravě by tedy dokument mohl vypadat takto:

```
<varianta1>
  <math>
    <mrow>
      <mrow>
        <mo>(</mo>
        <mrow>
          <msqrt>
            <mrow>
              <mi>x</mi>
              <mo>+</mo>
              <mn>1</mn>
            </mrow>
          </msqrt>
          <mo>+</mo>
          <mn>1</mn>
        </mrow>
        <mo>)</mo>
      </mrow>
    </mrow>
  </math>
</varianta1>
```

7.2.3 Příklad

Pro ukázku použiji jednoduchý příklad jak by měl vypadat XML dokument po použití matematických vzroců z programu MathType:

Kolik je $\frac{1}{3}+\frac{2}{7}$?

a) $\frac{1}{3}$, b) $\frac{2}{7}$, c) $\frac{13}{21}$, d) $\frac{1}{357}$

Obsah XML dokumentu, ze kterého bude vybírat webová stránka:

```
<?xml version="1.0"?>
<Root xmlns:m="http://www.w3.org/1998/Math/MathML">
  <priklad id="1">

    <zadani>Kolik je
      <math>
        <mrow>
          <mfrac>
            <mn>1</mn>
            <mn>3</mn>
          </mfrac>
          <mo>+</mo>
          <mfrac>
            <mn>2</mn>
            <mn>7</mn>
          </mfrac>
        </mrow>
      </math>?</zadani>

    <varianta1>
      <math>
        <mrow>
          <mfrac>
            <mn>1</mn>
            <mn>3</mn>
          </mfrac>
        </mrow>
      </math>
    </varianta1>

    <varianta2>
      <math>
```

```
        <mrow>
          <mfrac>
            <mn>2</mn>
            <mn>7</mn>
          </mfrac>
        </mrow>
      </math>
    </varianta2>

    <varianta3>
      <math>
        <mrow>
          <mfrac>
            <mrow>
              <mn>13</mn>
            </mrow>
            <mrow>
              <mn>21</mn>
            </mrow>
          </mfrac>
        </mrow>
      </math>
    </varianta3>

    <varianta4>
      <math>
        <mrow>
          <mfrac>
            <mn>1</mn>
            <mrow>
              <mn>357</mn>
            </mrow>
          </mfrac>
        </mrow>
      </math>
    </varianta4>

    <reseni>3</reseni>
  </priklad>
</Root>
```

Poznámka:

První řádek (<?xml version="1.0"?>) říká, která verze XML je použita.

Druhý řádek startuje XML soubor s atributem jmeného prostoru, který poukazuje na použití MathML jazyka.

7.3 Skript pro MathML v MSIE a Mozilla Firefox

Skript využitý pro správné zobrazování jazyka MathML v internetových prohlížečích MSIE a Mozilla Firefox.

```
<script>

function convertMath(node) // pro Gecko
{
 if (node.nodeType==1)
 {
  var newnode = document.createElementNS("http://www.w3.org/1998/Math/MathML",
        node.nodeName.toLowerCase());
  for(var i=0; i < node.attributes.length; i++)
        newnode.setAttribute(node.attributes[i].nodeName, node.attributes[i].nodeValue);
  for (i=0; i<node.childNodes.length; i++)
  {
   var st = node.childNodes[i].nodeValue;
   if (st==null || st.slice(0,1)!=" " && st.slice(0,1)!="\n")
         newnode.appendChild(convertMath(node.childNodes[i]));
  }
  return newnode;
 }
 else return node;
}

function convert()
{
 var mmlnode = document.getElementsByTagName("math");
 var st,str,node,newnode;
 for (var i=0; i<mmlnode.length; i++)
  if (document.createElementNS!=null)
        mmlnode[i].parentNode.replaceChild(convertMath(mmlnode[i]),mmlnode[i]);
  else  // konverze pro MSIE
  {
   str = "";
   node = mmlnode[i];
   while (node.nodeName!="/MATH")
   {
    st = node.nodeName.toLowerCase();
    if (st=="#text") str += node.nodeValue;
    else
    {
     str += (st.slice(0,1)=="/" ? "</m:"+st.slice(1) : "<m:"+st);
```

```
        if (st.slice(0,1)!="/")
          for(var j=0; j < node.attributes.length; j++)
            if (node.attributes[j].nodeValue!="italic" &&
              node.attributes[j].nodeValue!="" &&
              node.attributes[j].nodeValue!="inherit" &&
              node.attributes[j].nodeValue!=undefined)
             str += "    "+node.attributes[j].nodeName+"="+"\""+node.attributes[j].nodeValue+"\"";
        str += ">";
       }
       node = node.nextSibling;
       node.parentNode.removeChild(node.previousSibling);
      }
      str += "</m:math>";
      newnode = document.createElement("span");
      node.parentNode.replaceChild(newnode,node);
      newnode.innerHTML = str;
    }
}

// pridani pluginu MathPlayer do kazde webove stranky
if (document.createElementNS==null)
{
  document.write("<object id=\"mathplayer\"\
  classid=\"clsid:32F66A20-7614-11D4-BD11-00104BD3F987\"></object>");
  document.write("<?import namespace=\"m\" implementation=\"#mathplayer\"?>");
}

</script>
```